SpaceVim

A modular Vim/Neovim configuration

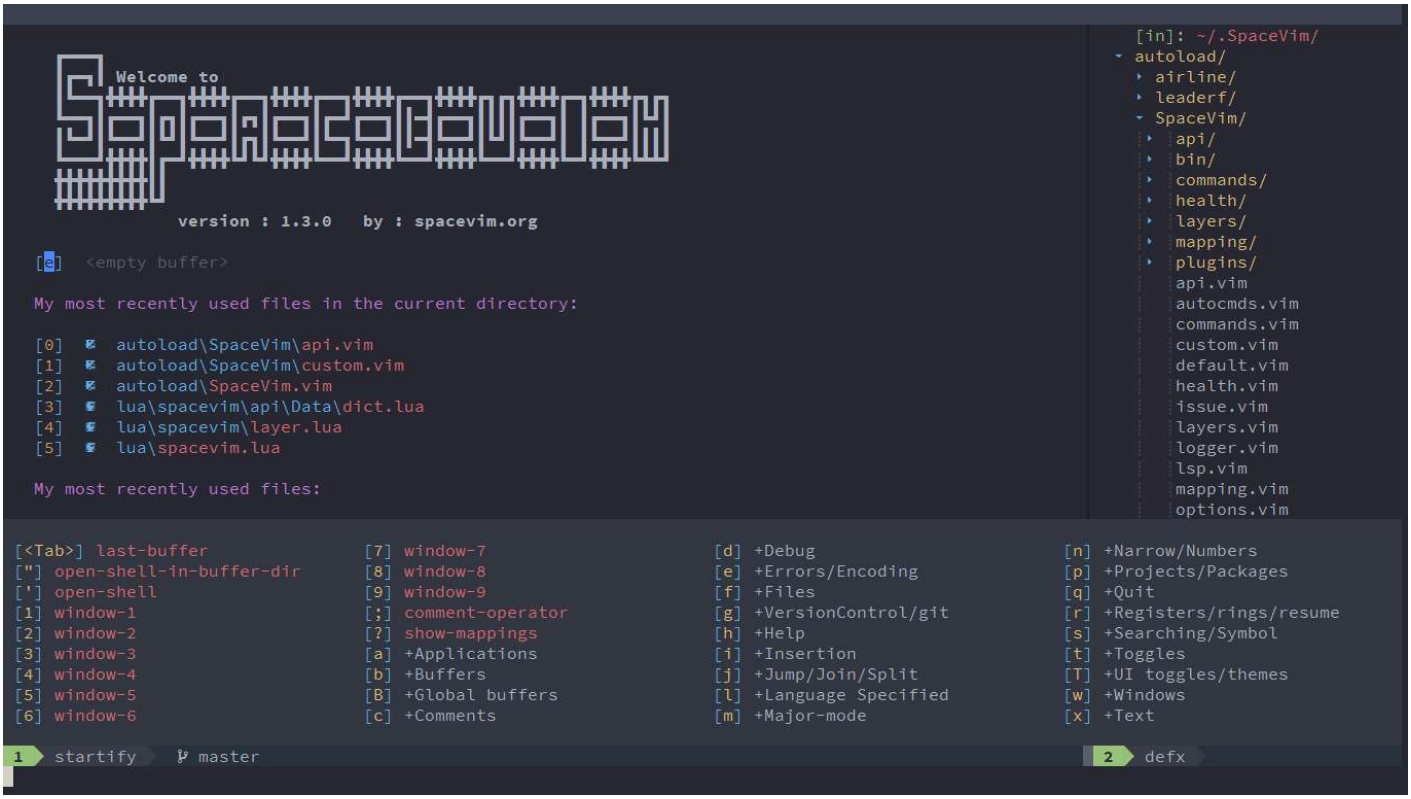
Documentation

Highlighted Features

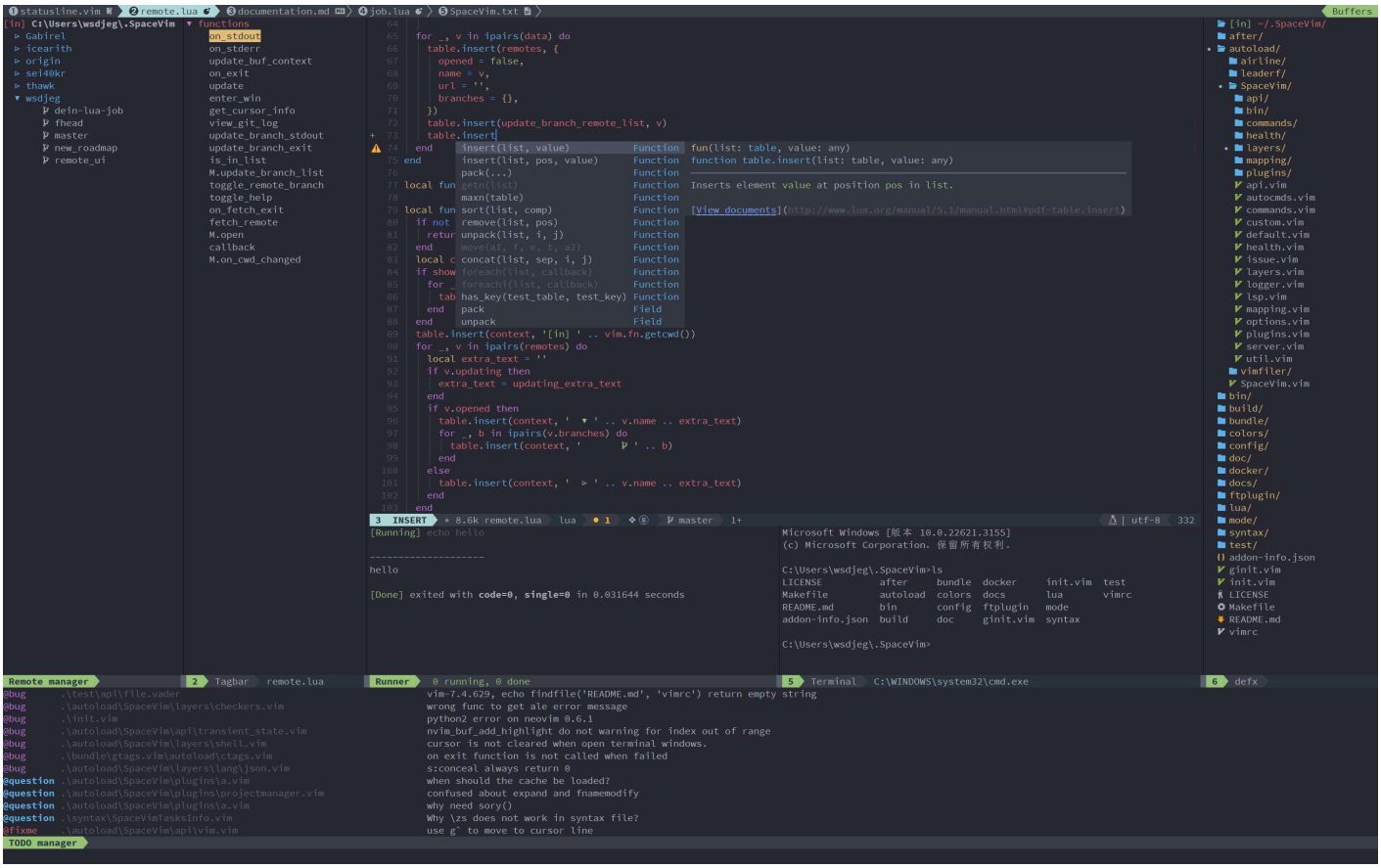
- **Modularization:** Plugins are organized in layers.
- **Compatible API:** A series of compatible API for Vim/Neovim.
- **Great documentation:** Everything is documented in `:h SpaceVim`.
- **Better experience:** Most of the core plugins have been rewritten using Lua.
- **Beautiful UI:** The interface has been carefully designed.
- **Mnemonic key bindings:** Key bindings are organized using mnemonic prefixes.
- **Lower the risk of RSI:** Heavily using the `<Space>` key instead of modifiers.

Screenshots

welcome page



workflow



- colorscheme: one
- windows: Git remotes, outline, Todos, Code runner, Terminal, file explore.
- code completion engine: nvim-cmp

New Concepts

Transient-states

SpaceVim defines a wide variety of transient states (temporary overlay maps) where it makes sense. This prevents one from doing repetitive and tedious presses on the SPC (space) key.

When a transient state is active, a documentation is displayed in the transient state buffer. Additional information may as well be displayed in it.

Move Text Transient State:

Update and Rollback

Update SpaceVim itself

There are several methods of updating the core files of SpaceVim. It is recommended to update the packages first; see the next section.

Automatic Updates

By default, this feature is disabled. It would slow down the startup of Vim/Neovim. If you like this feature, add the following to your custom configuration file.

```
[options]
    automatic_update = true
```

SpaceVim will automatically check for a new version every startup. You have to restart Vim after updating.

Updating from the SpaceVim Buffer

Users can use command `:SPUpdate SpaceVim` to update SpaceVim. This command will open a new buffer to show the process of updating.

Updating Manually with git

For users who prefer to use the command line, they can use the following command in a terminal to update SpaceVim manually:

```
git -C ~/.SpaceVim pull
```

Update plugins

Use `:SPUpdate` command to update all the plugins and SpaceVim itself. After `:SPUpdate`, you can assign plugins need to be updated. Use `Tab` to complete plugin names after `:SPUpdate`.

Reinstall plugins

When a plugin has failed to update or is broken, Use the `:SPReinstall` command to reinstall the plugin. The plugin's name can be completed via the key binding `<Tab>`.

For example:

```
:SPReinstall echodoc.vim
```

Get SpaceVim log

The runtime log of SpaceVim can be obtained via the key binding `SPC h L`. To get the debug information about the current SpaceVim environment, Use the command `:SPDebugInfo!`. This command will open a new buffer where default information will be shown. You can also use `SPC h I` to open a buffer with SpaceVim's issue template.

Custom Configuration

The very first time SpaceVim starts up, it will ask you to choose a mode, `basic mode` or `dark powered mode`. Then it will create a `SpaceVim.d/init.toml` in your `$HOME` directory. All the user configuration files are stored in `~/.SpaceVim.d/` directory.

`~/.SpaceVim.d/` will be added to `&runtimepath`.

It is also possible to override the location of `~/.SpaceVim.d/` using the environment variable `SPACEVIMDIR`. Of course, you can also use symlinks to change the location of this directory.

SpaceVim also supports project specific configuration files. The init file is `.SpaceVim.d/init.toml` in the root of your project. The local `.SpaceVim.d/` will also be added to the `&runtimepath`.

Please be aware that if there are errors in your `init.toml`, the setting will not be applied. See FAQ.

All SpaceVim options can be found in `:h SpaceVim-options`, the key is the same as the option name without the `g:spacevim_` prefix.

Comprehensive documentation is available in `:h SpaceVim`. Users can also use `SPC h SPC` to fuzzy find the documentation of SpaceVim options. This key binding requires one fuzzy finder layer to be loaded.

Add custom plugins

If you want to add plugins from GitHub, just add the repo name to the `custom_plugins` section:

```
[[custom_plugins]]
    repo = 'lilydjwg/colorizer'
    # `on_cmd` option means this plugin will be loaded
    # only when the specific commands are called.
    # for example, when `:ColorHighlight` or `:ColorToggle`
    # commands are called.
    on_cmd = ['ColorHighlight', 'ColorToggle']
    # `on_func` option means this plugin will be loaded
    # only when the specific functions are called.
    # for example, when `colorizer#ColorToggle()` function is called.
    on_func = 'colorizer#ColorToggle'
    # `merged` option is used for merging plugins directory.
    # When `merged` is `true`, all files in this custom plugin
```

```
    # will be merged into `~/.cache/vimfiles/.cache/init.vim/`
    # for neovim or `~/.cache/vimfiles/.cache/vimrc/` for vim.
    merged = false
    # For more options see `:h dein-options`.
```

You can also use the url of the repository, for example:

```
[[custom_plugins]]
    repo = "https://gitlab.com/code-stats/code-stats-vim.git"
    merged = false
```

For adding multiple custom plugins:

```
[[custom_plugins]]
    repo = 'lilydjwg/colorizer'
    merged = false

[[custom_plugins]]
    repo = 'joshdick/onedark.vim'
    merged = false
```

disable existing plugins

If you want to disable plugins which are added by SpaceVim, you can use SpaceVim `disabled_plugins` in the `[options]` section of your configuration file.

```
[options]
    # NOTE: the value should be a list, and each item is the name of the plugin.
    disabled_plugins = ["clighter", "clighter8"]
```

Bootstrap Functions

Due to the limitations of toml syntax, SpaceVim provides two bootstrap function options `bootstrap_before` and `bootstrap_after`, which specify two Vim custom functions.

To enable this feature you need to add the following config to the `[options]` section of your configuration file `~/.SpaceVim.d/init.toml`.

```
[options]
    bootstrap_before = 'myspacevim#before'
    bootstrap_after = 'myspacevim#after'
```

The difference is that the bootstrap before function will be called before SpaceVim core, and the bootstrap after function is called on autocmd `VimEnter`, so you can override defaults key bindings in `bootstrap_after` function.

The bootstrap functions should be placed in the `autoload` directory in `~/.SpaceVim.d/`. In our case, create file `~/.SpaceVim.d/autoload/myspacevim.vim` with the following contents, for example:

```
function! myspacevim#before() abort
    let g:neomake_c_enabled_makers = ['clang']
    " you can defined mappings in bootstrap function
    " for example, use kj to exit insert mode.
    inoremap kj <Esc>
endfunction

function! myspacevim#after() abort
    " you can remove key binding in bootstrap_after function
    " for example, remove F3 which is to open file tree by default.
    unmap <F3>
    " create new key binding to open file tree.
    nnoremap <silent> <F3> :Defx<Cr>
endfunction
```

Within the bootstrap function, you can also use `:lua` command. for example:

```
function! myspacevim#before() abort
    lua << EOF
    local opt = requires('spacevim.opt')
    opt.enable_projects_cache = false
    opt.enable_statusline_mode = true
EOF
endfunction
```

The `bootstrap_before` will be called after custom configuration file is loaded. And the `bootstrap_after` will be called after Vim Enter autocmd.

If you want to add custom SPC prefix key bindings, you can add them to bootstrap function, **make sure** the key bindings are not used in SpaceVim.

```
function! myspacevim#before() abort
    call SpaceVim#custom#SPCGroupName(['G'], '+TestGroup')
    call SpaceVim#custom#SPC('nore', ['G', 't'], 'echom 1', 'echomessage 1', 1)
endfunction
```

Similarly, if you want to add custom key bindings prefixed by language leader key, which is typically `,`, you can add them to the bootstrap function. **Make sure** that the key bindings are not used by SpaceVim.

```
function! myspacevim#before() abort
    call SpaceVim#custom#LangSPCGroupName('python', ['G'], '+TestGroup')
    call SpaceVim#custom#LangSPC('python', 'nore', ['G', 't'], 'echom 1', 'echome
endfunction
```

Vim compatible mode

The different key bindings between SpaceVim and vim are shown as below.

- In vim the `s` key replaces the character under the cursor. In SpaceVim it is the `Window` key binding's specific leader in **Normal** mode. This leader can be changed via the `windows_leader` option which uses `s` as the default variable. If you still prefer the original function of `s`, you can use an empty string to disable this feature.

```
[options]
    windows_leader = ''
```

- In vim the `,` key repeats the last `f`, `F`, `t` and `T`, but in SpaceVim it is the language specific Leader key. To disable this feature, set the option `enable_language_specific_leader` to `false` in the `[options]` section of your configuration file.

```
[options]
    enable_language_specific_leader = false
```

- In vim the `q` key does recording, but in SpaceVim it is used to close current window. The option for setting the key binding to close the current window is `windows_smartclose`, and the default value is `q`. If you prefer to use the original function of `q`, you can use an empty string to disable this feature.

```
[options]
    windows_smartclose = ''
```

- In SpaceVim the `jk` key (press `j` then `k` in succession) has been mapped to `<Esc>` in insert mode. To disable this key binding, set `escape_key_binding` to an empty string.

```
[options]
    escape_key_binding = ''
```

- In vim the `Ctrl-a` binding on the command line can auto-complete variable names, but in SpaceVim it moves to the cursor to the beginning of the command line.
- In SpaceVim the `Ctrl-b` binding on the command line is mapped to `<Left>`, which will move cursor to the left.
- In SpaceVim the `Ctrl-f` binding on the command line is mapped to `<Right>`, which will move cursor to the right.

SpaceVim provides a vimcompatible mode, in vimcompatible mode, all the differences above will disappear. You can enable the vimcompatible mode by adding `vimcompatible = true` to the `[options]` section of your configuration file.

If you want to disable any differences above, use the relevant options. For example, in order to disable language specific leader, you may add the following lines to the `[options]` section of `~/.SpaceVim.d/init.toml`:

```
[options]
    enable_language_specific_leader = false
```

Send a PR to add the differences you found in this section.

Private Layers

This section is an overview of layers. A more extensive introduction to writing configuration layers can be found in SpaceVim's layers page (recommended reading!).

Purpose

Layers help collect related packages together to provide features. For example, the `lang#python` layer provides auto-completion, syntax checking, and REPL support for python files. This approach helps keep configurations organized and reduces overhead for users by keeping them from having to think about what packages to install. To install all the `python` features users only need to add the `lang#python` layer to their custom configuration file.

Structure

In SpaceVim, a layer is a single file. In a layer, for example, `autocomplete` layer, the file is `autoload/SpaceVim/layers/autocomplete.vim`, and there are three public functions:

- `SpaceVim#layers#autocomplete#plugins()`: returns a list of the plugins used by this plugin
- `SpaceVim#layers#autocomplete#config()`: The layer's configuration, such as key bindings and autocmds
- `SpaceVim#layers#autocomplete#set_variable()`: Function for setting layer options
- `SpaceVim#layers#autocomplete#get_options()`: Returns a list of all the available layer options

Debug upstream plugins

If you found out that one of the built-in plugins has bugs, and you want to debug it, You can follow these steps:

1. Disable the plugin Take disabling neomake.vim for instance:

```
[options]
    disabled_plugins = ["neomake.vim"]
```

1. Add a forked plugin or add a local plugin Use the toml file to load custom plugins:

```
[[custom_plugins]]
    repo = "wsdjeg/neomake.vim"
    # note: you need to disable merged feature
    merged = false
```

Use the `bootstrap_before` function to add the local plugin:

```
function! myspacevim#before() abort
    set rtp+=~/path/to/your/localplugin
endfunction
```

Interface elements

SpaceVim has a minimalistic and distraction free UI:

- custom airline with color feedback according to current check status
- custom icon in sign column and error feedbacks for checker.

Colorschemes

The default colorscheme of SpaceVim is gruvbox. There are two variants of this colorscheme, dark and light. Some aspects of these colorschemes can be customized in the custom configuration file, read `:h gruvbox`.

It is possible to change the colorscheme in `~/.SpaceVim.d/init.toml` with the variable `colorscheme`. For instance, to specify `desert` add the following to the `[options]` section:

```
[options]
    colorscheme = "desert"
    colorscheme_bg = "dark"
```

Mappings	Descriptions
SPC T n	switch to a random colorscheme listed in colorscheme layer.
SPC T s	select a theme using a fuzzy finder.

All the included colorschemes can be found in colorscheme layer.

SpaceVim supports true colors in terminal, and it is disabled by default, to enable this feature, you should make sure your terminal supports true colors. For more information see: Colours in terminal.

If your terminal does not support true colors, you can disable SpaceVim true colors feature in `[options]` section:

```
enable_guicolors = false
```

Font

The default font used by SpaceVim is Sauce Code Nerd Font. It is recommended to install it on your system if you wish to use it.

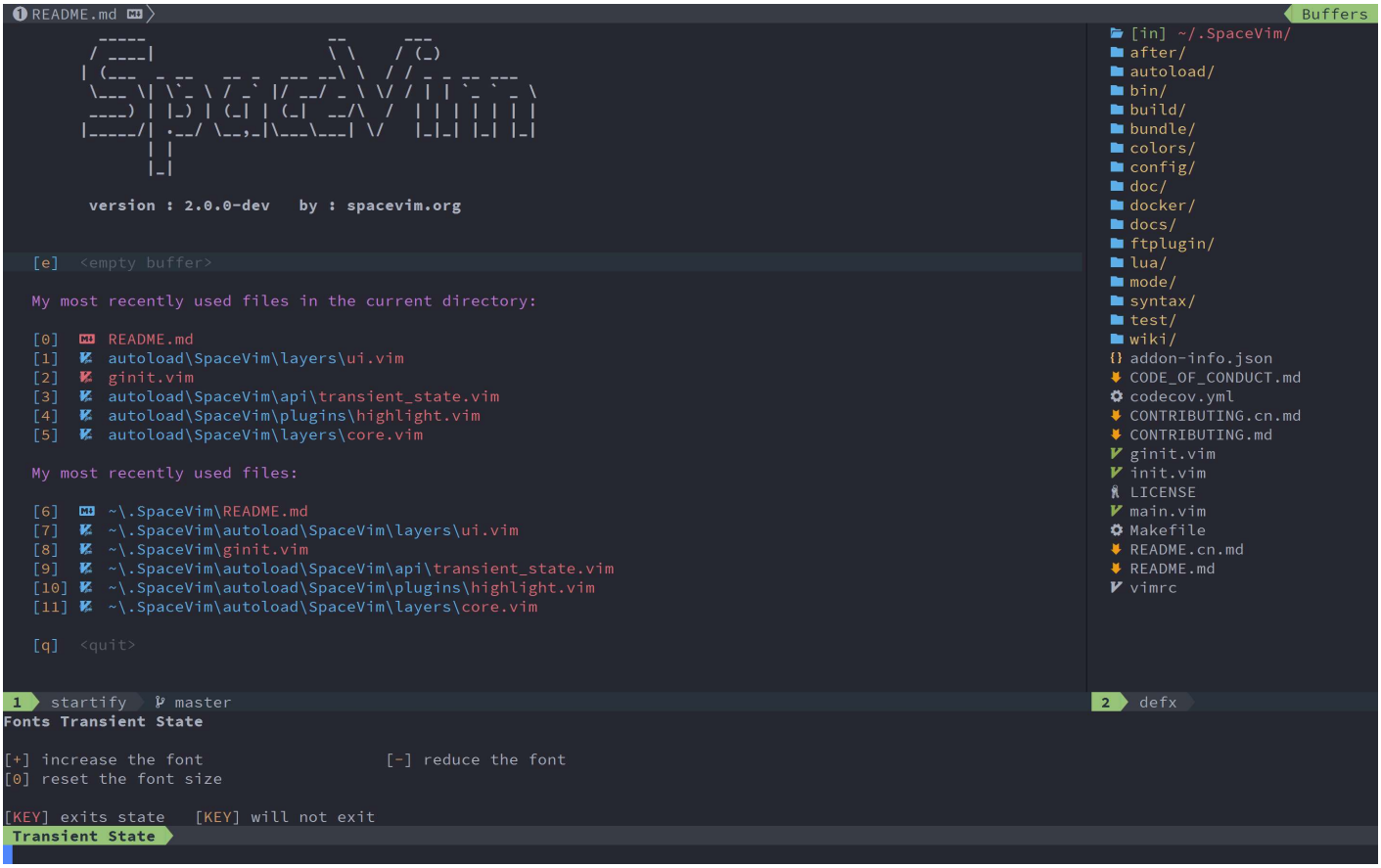
To change the default font set the variable `guifont` in your `~/.SpaceVim.d/init.toml` file. By default its value is:

```
[options]
    guifont = "SauceCodePro Nerd Font Mono:h11"
```

If the specified font is not found, the fallback one will be used (depends on your system). Also note that changing this value has no effect if you are running Vim/Neovim in terminal.

Increase/Decrease fonts

Key Bindings	Descriptions
SPC z .	open font transient state



In font transient state:

Key Bindings	Descriptions
+	increase the font size
-	decrease the font size
Any other key	leave the transient state

Mouse

Mouse support is enabled in Normal mode and Visual mode by default. To change the default value, you need to use the bootstrap function.

For example, to disable mouse:

```
function! myspacevim#before() abort
    set mouse=
endfunction
```

Read `:h 'mouse'` for more info.

Scrollbar

The scrollbar requires floating window of neovim or popup of vim8. It is disabled by default. To enable the scrollbar, you need to change `enable_scrollbar` option in ui layer.

```
[[layers]]
  name = "ui"
  enable_scrollbar = true
```

UI Toggles

Some UI indicators can be toggled on and off (toggles start with t and T):

Key Bindings	Descriptions
`SPC t 8`	highlight characters past the 80th column
`SPC t a`	toggle autocomplete (only available with `autocomplete_method = deoplete`)
`SPC t f`	display the fill column (by default `max_column` is 120)
`SPC t h h`	toggle highlight of the current line
`SPC t h i`	toggle highlight indentation levels
`SPC t h c`	toggle highlight current column
`SPC t h s`	toggle syntax highlighting
`SPC t i`	toggle indentation guide at point
`SPC t n`	toggle line numbers
`SPC t b`	toggle background
`SPC t c`	toggle conceal
`SPC t p`	toggle paste mode
`SPC t P`	toggle auto parens mode
`SPC t t`	open tabs manager
`SPC T ~`	display ~ in the fringe on empty lines
`SPC T F` / `F11`	toggle frame fullscreen
`SPC T f`	toggle display of the fringe
`SPC T m`	toggle menu bar
`SPC T t`	toggle tool bar

Statusline

The `core#statusline` layer provides a heavily customized powerline with the following capabilities:

- show the window number
- show the current mode
- color code for current state

- show the index of search results
- toggle syntax checking info
- toggle battery info
- toggle major mode lighters
- show VCS information (branch, hunk summary) (requires `git` and `VersionControl` layers)

Key Bindings	Descriptions
`SPC [1-9]`	jump to the windows with the specific number

Reminder of the color codes for the states:

Mode	Color
Normal	Grey
Insert	Blue
Visual	Orange
Replace	Aqua

All the colors are based on the current colorscheme.

Some elements can be dynamically toggled:

Key Bindings	Descriptions
`SPC t m b`	toggle the battery status (need to install acpi)
`SPC t m c`	toggle the org task clock (available in org layer)(TODO)
`SPC t m i`	toggle the input method
`SPC t m m`	toggle the major mode lighters
`SPC t m M`	toggle the filetype section
`SPC t m n`	toggle the cat! (If colors layer is declared in your dotfile)(TODO)
`SPC t m p`	toggle the cursor position
`SPC t m t`	toggle the time
`SPC t m d`	toggle the date
`SPC t m T`	toggle the mode line itself
`SPC t m v`	toggle the version control info

nerd font installation:

By default SpaceVim uses nerd-fonts, which can be downloaded from their website.

syntax checking integration:

When syntax checking major mode is enabled, a new element appears showing the number of errors and warnings.

The default highlight group and colors are:

highlight group	color
SpaceVim_statusline_error	#ffc0b9
SpaceVim_statusline_warn	#fce094
SpaceVim_statusline_info	#8cf8f7
SpaceVim_statusline_hint	#a6dbff

Search index integration:

Search index shows the number of occurrences when performing a search via `/` or `?`. SpaceVim integrates the search status nicely by displaying it temporarily when `n` or `N` are being pressed. See the 20/22 segment in the screenshot below.

Search index is provided by `incsearch` layer, to enable this layer:

```
[[layers]]
    name = "incsearch"
```

Battery status integration:

acpi displays the remaining battery percentage as well as the time remaining to charge or discharge the battery completely.

A color code is used for the battery status:

Battery State	Color
Charging	Green
Discharging	Orange
Critical	Red

All the colors are based on the current colorscheme.

Statusline separators:

It is possible to easily customize the statusline separator by setting the `statusline_separator` variable in your custom configuration file and then redraw the statusline. For instance, if you want to set the separator back to the well-known arrow separator, add the following snippet to the `[options]` section of your configuration file:

```
[options]
    statusline_separator = 'arrow'
```

Here is an exhaustive set of screenshots for all the available separators:

Separator	Screenshot
arrow	
curve	
slant	

Separator	Screenshot
`nil`	
`fire`	

major modes:

The major mode area can be toggled on and off with `SPC t m m`.

Unicode symbols are displayed by default. Add `statusline_unicode = false` to your custom configuration file to use ASCII characters instead (may be useful in the terminal if you cannot set an appropriate font).

The letters displayed in the statusline correspond to the key bindings used to toggle them.

Key Bindings	Unicode	ASCII	Mode
`SPC t 8`	⑧	8	toggle character highlighting for long lines
`SPC t f`	ⓕ	f	fill-column-indicator mode
`SPC t s`	ⓢ	s	syntax checking (neomake)
`SPC t S`	Ⓢ	S	enabled in spell checking
`SPC t w`	ⓦ	w	whitespace mode (highlight trailing whitespace)
`SPC t W`	Ⓦ	W	wrap line mode

The status of major mode will be cached, the cache will be loaded when spacevim startup. If you want to disable major mode cache, you need to charge the layer option of `core#statusline` layer.

```
[[layers]]
  name = 'core#statusline'
  major_mode_cache = false
```

colorscheme of statusline:

By default SpaceVim only supports colorschemes included in colorscheme layer.

If you want to contribute a theme please check the template of a statusline theme.

```
" the theme colors should be
" [
"     \ [ a_guifg,  a_guibg,  a_ctermfg,  a_ctermbg],
"     \ [ b_guifg,  b_guibg,  b_ctermfg,  b_ctermbg],
"     \ [ c_guifg,  c_guibg,  c_ctermfg,  c_ctermbg],
"     \ [ z_guibg,  z_ctermbg],
"     \ [ i_guifg,  i_guibg,  i_ctermfg,  i_ctermbg],
"     \ [ v_guifg,  v_guibg,  v_ctermfg,  v_ctermbg],
"     \ [ r_guifg,  r_guibg,  r_ctermfg,  r_ctermbg],
"     \ [ ii_guifg, ii_guibg, ii_ctermfg, ii_ctermbg],
"     \ [ in_guifg, in_guibg, in_ctermfg, in_ctermbg],
" \ ]
" group_a: window id
```

```
" group_b/group_c: stausline sections
" group_z: empty area
" group_i: window id in insert mode
" group_v: window id in visual mode
" group_r: window id in select mode
" group_ii: window id in iedit-insert mode
" group_in: windows id in iedit-normal mode
function! SpaceVim#mapping#guide#theme#gruvbox#palette() abort
    return [
    \ ['#282828', '#a89984', 246, 235],
    \ ['#a89984', '#504945', 239, 246],
    \ ['#a89984', '#3c3836', 237, 246],
    \ ['#665c54', 241],
    \ ['#282828', '#83a598', 235, 109],
    \ ['#282828', '#fe8019', 235, 208],
    \ ['#282828', '#8ec07c', 235, 108],
    \ ['#282828', '#689d6a', 235, 72],
    \ ['#282828', '#8f3f71', 235, 132],
    \ ]
endfunction
```

This example is the gruvbox colorscheme, if you want to use same colors when switching between different colorschemes, you may need to set `custom_color_palette` in the `[options]` section of your custom configuration file. For example:

```
[options]
    custom_color_palette = [
        ["#282828", "#a89984", 246, 235],
        ["#a89984", "#504945", 239, 246],
        ["#a89984", "#3c3836", 237, 246],
        ["#665c54", 241],
        ["#282828", "#83a598", 235, 109],
        ["#282828", "#fe8019", 235, 208],
        ["#282828", "#8ec07c", 235, 108],
        ["#282828", "#689d6a", 235, 72],
        ["#282828", "#8f3f71", 235, 132],
    ]
```

Custom section

You can use the bootstrap function to add a custom section to the statusline, for example:

```
function! s:test_section() abort
  return 'ok'
endfunction
call SpaceVim#layers#core#statusline#register_sections('test', function('s:test_s
```

Then, add `test` section to `statusline_right_sections` option:

```
[options]
    statusline_right_sections = ['cursorpos', 'percentage', 'test']
```

Tabline

Buffers will be listed on the tabline if there is only one tab, each item contains the index, buffer name and the filetype icon. If there is more than one tab, all of them will be listed on the tabline. Each item can be quickly accessed by using `<Leader> number`. Default `<Leader>` is `\`.

Key Bindings	Descriptions
`<Leader> 1`	Jump to index 1 on tabline
`<Leader> 2`	Jump to index 2 on tabline
`<Leader> 3`	Jump to index 3 on tabline
`<Leader> 4`	Jump to index 4 on tabline
`<Leader> 5`	Jump to index 5 on tabline
`<Leader> 6`	Jump to index 6 on tabline
`<Leader> 7`	Jump to index 7 on tabline
`<Leader> 8`	Jump to index 8 on tabline
`<Leader> 9`	Jump to index 9 on tabline
`g r`	Switch to alternate tab (switch back and forth)

The following two key bindings require neovim v0.10.0+.

Key Bindings	Descriptions
`Ctrl-Shift-Right`	move current buffer to the right
`Ctrl-Shift-Left`	move current buffer to the left

Note: `SPC Tab` is the key binding for switching to alternate buffer. Read Buffers and Files section for more info.

SpaceVim tabline also supports mouse click, the left mouse button will switch to the buffer, while the middle mouse button will delete the buffer.

NOTE: This feature is only supported in Neovim with `has('tablineat')`.

Key Bindings	Descriptions
`<Mouse-left>`	Switch to the buffer
`<Mouse-middle>`	Delete the buffer

Tab manager:

You can also use `SPC t t` to open the tab manager window.

Key bindings within the tab manager window:

Key Bindings	Descriptions
`o`	Close or expand tab windows.
`r`	Rename the tab under the cursor.
`n`	Create new named tab below the cursor tab
`N`	Create new tab below the cursor tab
`x`	Delete the tab
`Ctrl-Shift-Up`	Move tab backward
`Ctrl-Shift-Down`	Move tab forward
`<Enter>`	Switch to the window under the cursor.

File tree

SpaceVim uses `nerdtree` as the default file tree, the default key binding is `<F3>`. SpaceVim also provides `SPC f t` and `SPC f T` to open the file tree.

To change the filemanager plugin insert the following to the `[options]` section of your configuration file.

```
[options]
    # file manager plugins supported in SpaceVim:
    # - nerdtree (default)
    # - vimfiler: you need to build the vimproc.vim in bundle/vimproc.vim directo
    # - defx: requires +py3 feature
    # - neo-tree: require neovim 0.7.0
    filemanager = "nerdtree"
```

VCS integration is supported, there will be a column status, this feature may make filetree slow, so it is not enabled by default. To enable this feature, add `enable_filetree_gitstatus = true` to your custom configuration file. Here is a picture of this feature:

There is also an option to configure show/hide the file tree, default to show. To hide the file tree by default, you can use the `enable_vimfiler_welcome` in the `[options]` section:

```
[options]
    enable_vimfiler_welcome = false
```

There is also an option to configure the side of the file tree, by default it is right. To move the file tree to the left, you can use the `filetree_direction` option:

```
[options]
    filetree_direction = "left"
```

File tree navigation

Navigation is centered on the `hjkl` keys with the hope of providing a fast navigation experience like in vifm:

Key Bindings	Descriptions
`<F3>` / `SPC f t`	Toggle file explorer
with in file tree	
`<Left>` / `h`	go to parent node and collapse expanded directory
`<Down>` / `j`	select next file or directory
`<Up>` / `k`	select previous file or directory
`<Right>` / `l`	open selected file or expand directory

Key Bindings	Descriptions
`<Enter>`	open file or switch to directory
`N`	Create new file under cursor
`r`	Rename the file under cursor
`d`	Delete the file under cursor
`K`	Create new directory under cursor
`y y`	Copy file full path to system clipboard
`y Y`	Copy file to system clipboard
`P`	Paste file to the position under the cursor
`.`	Toggle hidden files
`s v`	Split edit
`s g`	Vertical split edit
`p`	Preview
`i`	Switch to directory history
`v`	Quick look
`g x`	Execute with vimfiler associated
`'`	Toggle mark current line
`V`	Clear all marks
`>`	increase filetree screenwidth
`<`	decrease filetree screenwidth
`<Home>`	Jump to first line
`<End>`	Jump to last line
`Ctrl-h`	Switch to project root directory
`Ctrl-r`	Redraw

Open file with file tree.

If only one file buffer is opened, a file is opened in the active window, otherwise we need to use vim-choosewin to select a window to open the file.

Key Bindings	Descriptions
`l` / `<Enter>`	open file in one window
`s g`	open file in a vertically split window

Key Bindings	Descriptions
`s v`	open file in a horizontally split window

Override filetree key bindings

If you want to override the default key bindings in filetree windows. You can use User autocmd in bootstrap function. for examples:

```
function! myspacevim#before() abort
    autocmd User NerdTreeInit
        \ nnoremap <silent><buffer> <CR> :<C-u>call
        \ g:NERDTreeKeyMap.Invoke('o')<CR>
endfunction
```

Here is all the autocmd for filetree:

- nerdtree: `User NerdTreeInit`
- defx: `User DefxInit`
- vimfiler: `User VimfilerInit`

General usage

The following key bindings are the general key bindings for moving the cursor.

Key Bindings	Descriptions
`h`	move cursor left
`j`	move cursor down
`k`	move cursor up
`l`	move cursor right
`<Up>`, `<Down>`	Smart up and down
`H`	move cursor to the top of the screen
`L`	move cursor to the bottom of the screen
`<`	Indent to left and re-select
`>`	Indent to right and re-select
`}`	paragraphs forward
`{`	paragraphs backward
`Ctrl-f` / `Shift-Down` / `<PageDown>`	Smooth scrolling forwards
`Ctrl-b` / `Shift-Up` / `<PageUp>`	Smooth scrolling backwards
`Ctrl-d`	Smooth scrolling downwards

Key Bindings	Descriptions
`Ctrl-u`	Smooth scrolling upwards
`Ctrl-e`	Smart scroll down (`3 Ctrl-e/j`)
`Ctrl-y`	Smart scroll up (`3Ctrl-y/k`)

Native functions

When vimcompatible is not enabled, some native key bindings of vim has been overrided. To use them, SpaceVim provides alternate key bindings:

Key bindings	Mode	Action
`<Leader> q r`	Normal	Same as native `q`
`<Leader> q r /`	Normal	Same as native `q /`, open cmdwin
`<Leader> q r ?`	Normal	Same as native `q ?`, open cmdwin
`<Leader> q r :`	Normal	Same as native `q :`, open cmdwin

Command line mode key bindings

After pressing `:`, you can switch to command line mode, here is a list of key bindings can be used in command line mode:

Key bindings	Descriptions
`Ctrl-a`	move cursor to beginning
`Ctrl-b`	Move cursor backward in command line
`Ctrl-f`	Move cursor forward in command line
`Ctrl-w`	delete a whole word
`Ctrl-u`	remove all text before cursor
`Ctrl-k`	remove all text after cursor
`Ctrl-c`/`Esc`	cancel command line mode
`Tab`	next item in popup menu
`Shift-Tab`	previous item in popup menu

Mappings guide

The mapping guide windows will be opened each time the prefix key is pressed in normal/visual mode. It will list all available key bindings and the short descriptions. The prefix can be `[SPC]`, `[WIN]` or `<Leader>`.

The prefixes are mapped to the following keys by default:

Prefix name	Custom options and default values	Descriptions
[SPC]	NONE / <Space>	default mapping prefix of SpaceVim
[WIN]	windows_leader / s	window mapping prefix of SpaceVim
<Leader>	default vim leader	default leader prefix of vim/Neovim

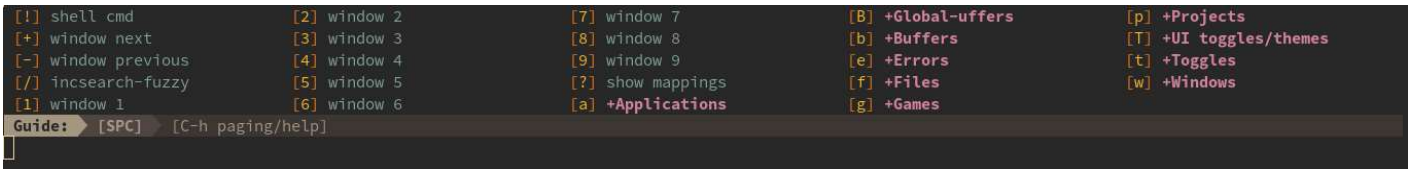
The default value of <Leader> is \, if you want to change this key, you need to use the bootstrap function. For example, to use , as the <Leader> key:

```
function! myspacevim#before() abort
    let g:mapleader = ','
endfunction
```

NOTE: When modifying the variable g:mapleader in a function. you can not omit the variable's scope. Because the default scope of a variable in function is l:. It seems different from what you see in vim help :h mapleader.

By default the guide buffer will be displayed 1000ms after the keys being pressed. You can change the delay by adding vim option 'timeoutlen' to your bootstrap function.

For example, after pressing <Space> in normal mode, you will see:



This guide shows you all the available key bindings that begin with [SPC], you can type b for all the buffer mappings, p for project mappings, etc.

After pressing Ctrl-h in guide buffer, you will get paging and help info in the statusline.

Keys	Descriptions
u	undo pressing
n	next page of guide buffer
p	previous page of guide buffer

Use SpaceVim#custom#SPC() to define custom SPC mappings. For instance:

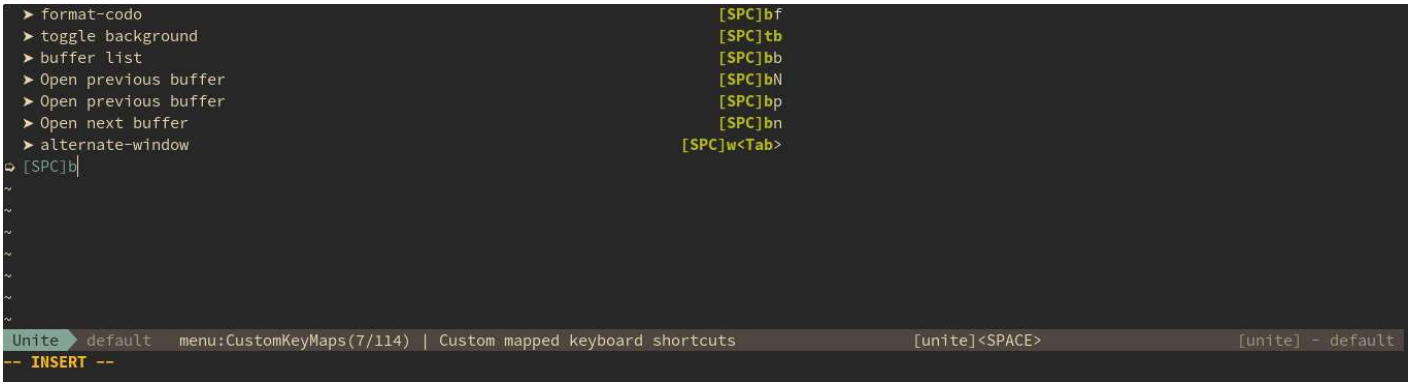
```
call SpaceVim#custom#SPC('nnoremap', ['f', 't'], 'echom "hello world"', 'test cus
```

The first parameter sets the type of shortcut key, which can be nnoremap or nmap, the second parameter is a list of keys, and the third parameter is an ex command or key binding, depending on whether the last parameter is true. The fourth parameter is a short description of this custom key binding.

Fuzzy find key bindings

It is possible to search for specific key bindings by pressing ? in the root of the guide buffer.

To narrow the list down, just insert the mapping keys or descriptions of what mappings you want, Unite/Denite will fuzzy find the mappings, to find buffer related mappings:



Then use <Tab> or <Up> and <Down> to select the mapping, press <Enter> to execute that command.

Mapping guide theme:

The default mapping guide theme is `leaderguide`, which is same as vim-leaderguide, there is alse another available theme called `whichkey`. To set the mapping guide theme, use following snippet:

```
[options]
    # the value can be `leaderguide` or `whichkey`
    leader_guide_theme = 'whichkey'
```

Editing

Moving text

Key	Action
> / Tab	Indent to right and re-select
< / Shift-Tab	Indent to left and re-select
Ctrl-Shift-Up	move lines up
Ctrl-Shift-Down	move lines down

Code indentation

The default indentation of code is 2, which is controlled by the option `default_indent`. If you prefer to use 4 as code indentation. Just add the following snippet to the `[options]` section in the SpaceVim's configuration file:

```
[options]
    default_indent = 4
```

The `default_indent` option will be applied to vim's `&tabstop`, `&softtabstop` and `&shiftwidth` options. By default, when the user inserts a <Tab>, it will be expanded to spaces. This feature can be disabled by `expand_tab` option the `[options]` section:

```
[options]
    default_indent = 4
```

```
    expand_tab = true
```

Text manipulation commands

Text related commands (start with `x`):

Key Bindings	Descriptions
`SPC x a #`	align region at #
`SPC x a %`	align region at %
`SPC x a &`	align region at &
`SPC x a (`	align region at (
`SPC x a )`	align region at)
`SPC x a [`	align region at [
`SPC x a ]`	align region at]
`SPC x a {`	align region at {
`SPC x a }`	align region at }
`SPC x a ,`	align region at ,
`SPC x a .`	align region at . (for numeric tables)
`SPC x a :`	align region at :
`SPC x a ;`	align region at ;
`SPC x a =`	align region at =
`SPC x a ¦`	align region at ¦
`SPC x a <Bar>`	align region at |
`SPC x a SPC`	align region at [SPC]
`SPC x a a`	align region (or guessed section) using default rules (TODO)
`SPC x a c`	align current indentation region using default rules (TODO)
`SPC x a l`	left-align with evil-lion (TODO)
`SPC x a L`	right-align with evil-lion (TODO)
`SPC x a r`	align region at user-specified regexp
`SPC x a o`	align region at operators `+-*/` etc
`SPC x c`	count the number of chars/words/lines in the selection region
`SPC x d w`	delete trailing whitespace

Key Bindings	Descriptions
SPC x d SPC	Delete all spaces and tabs around point, leaving one space
SPC x g l	set languages used by translate commands (TODO)
SPC x g t	translate current word using Google Translate
SPC x g T	reverse source and target languages (TODO)
SPC x i c	change symbol style to `lowerCamelCase`
SPC x i C	change symbol style to `UpperCamelCase`
SPC x i i	cycle symbol naming styles (i to keep cycling)
SPC x i -	change symbol style to `kebab-case`
SPC x i k	change symbol style to `kebab-case`
SPC x i _	change symbol style to `under_score`
SPC x i u	change symbol style to `under_score`
SPC x i U	change symbol style to `UP_CASE`
SPC x j c	set the justification to center
SPC x j f	set the justification to full (TODO)
SPC x j l	set the justification to left
SPC x j n	set the justification to none (TODO)
SPC x j r	set the justification to right
SPC x J	move down a line of text (enter transient state)
SPC x K	move up a line of text (enter transient state)
SPC x l d	duplicate a line or region
SPC x l r	reverse lines
SPC x l s	sort lines (ignorecase)
SPC x l S	sort lines (case-senstive)
SPC x l u	uniquify lines (ignorecase)
SPC x l U	uniquify lines (case-senstive)
SPC x o	use avy to select a link in the frame and open it (TODO)
SPC x O	use avy to select multiple links in the frame and open them (TODO)
SPC x t c	swap (transpose) the current character with the previous one
SPC x t C	swap (transpose) the current character with the next one

Key Bindings	Descriptions
`SPC x t w`	swap (transpose) the current word with the previous one
`SPC x t W`	swap (transpose) the current word with the next one
`SPC x t l`	swap (transpose) the current line with the previous one
`SPC x t L`	swap (transpose) the current line with the next one
`SPC x u`	lowercase text
`SPC x U`	uppercase text
`SPC x ~`	toggle case text
`SPC x w c`	count the words in the select region
`SPC x w d`	show dictionary entry of word from wordnik.com (TODO)
`SPC x <Tab>`	indent or dedent a region rigidly (TODO)

Text insertion commands

Text insertion commands (start with `i`):

Key bindings	Descriptions
`SPC i l l`	insert lorem-ipsum list
`SPC i l p`	insert lorem-ipsum paragraph
`SPC i l s`	insert lorem-ipsum sentence
`SPC i p 1`	insert simple password
`SPC i p 2`	insert stronger password
`SPC i p 3`	insert password for paranoids
`SPC i p p`	insert a phonetically easy password
`SPC i p n`	insert a numerical password
`SPC i u`	Search for Unicode characters and insert them into the active buffer.
`SPC i U 1`	insert UUIDv1 (use universal argument to insert with CID format)
`SPC i U 4`	insert UUIDv4 (use universal argument to insert with CID format)
`SPC i U U`	insert UUIDv4 (use universal argument to insert with CID format)

Tip: You can specify the number of password characters using a prefix argument (i.e. `10 SPC i p 1` will generate a 10 character simple password).

Expand regions of text

Key bindings available in visual mode:

Key bindings	Descriptions
v	expand visual selection of text to larger region
V	shrink visual selection of text to smaller region

Increase/Decrease numbers

Key Bindings	Descriptions
SPC n +	increase the number under the cursor by one and initiate transient state
SPC n -	decrease the number under the cursor by one and initiate transient state

In transient state:

Key Bindings	Descriptions
+	increase the number under the cursor by one
-	decrease the number under the cursor by one
Any other key	leave the transient state

Tip: You can set the step (1 by default) by using a prefix argument (i.e. 10 SPC n + will add 10 to the number under the cursor).

Copy and paste

If has('unnamedplus'), the register used by <Leader> y is +, otherwise it is *. Read :h registers for more info about other registers.

Key	Descriptions
<Leader> y	Copy selected text to system clipboard
<Leader> p	Paste text from system clipboard after here
<Leader> P	Paste text from system clipboard before here
<Leader> Y	Copy selected text to pastebin

To change the command of clipboard, you need to use bootstrap after function:

```
" for example, to use tmux clipboard:
function! myspacevim#after() abort
    call clipboard#set('tmux load-buffer -', 'tmux save-buffer -')
endfunction
```

within the runtime log (SPC h L), the clipboard command will be displayed:

```
[ clipboard ] [11:00:35] [670.246] [ Info  ] yank_cmd is:'tmux load-buffer -'
[ clipboard ] [11:00:35] [670.246] [ Info  ] paste_cmd is:'tmux save-buffer -'
```

The `<Leader> Y` key binding will copy selected text to a pastebin server. It requires `curl` in your `$PATH`. The default command is:

```
curl -s -F "content=<-" http://dpaste.com/api/v2/
```

This command will read stdin and copy it to dpaste server. It is same as:

```
echo "selected text" | curl -s -F "content=<-" http://dpaste.com/api/v2/
```

Commenting

Comments are handled by nerdcommenter, it's bound to the following keys.

Key Bindings	Descriptions
`SPC ;`	comment operator
`SPC c a`	switch to the alternative set of delimiters
`SPC c h`	hide/show comments
`SPC c l`	toggle line comments
`SPC c L`	comment lines
`SPC c u`	uncomment lines
`SPC c p`	toggle paragraph comments
`SPC c P`	comment paragraphs
`SPC c s`	comment with pretty layout
`SPC c t`	toggle comment on line
`SPC c T`	comment the line under the cursor
`SPC c y`	toggle comment and yank
`SPC c Y`	yank and comment
`SPC c $`	comment current line from cursor to the end of the line

Tip: `SPC ;` will start operator mode, in this mode, you can use a motion command to comment lines. For example, `SPC ; 4 j` will comment the current line and the following 4 lines.

Undo tree

Undo tree visualizes the undo history and makes it easier to browse and switch between different undo branches. The default key binding is `F7`. If `+python` or `+python3` is enabled, mundo will be loaded, otherwise undotree will be

loaded.

Key bindings within undo tree windows:

key bindings	description
G	move bottom
J	move older write
K	move newer write
N	previous match
P	play to
<2-LeftMouse>	mouse click
/	search
<CR>	preview
d	diff
<down>	move older
<up>	move newer
i	toggle inline
j	move older
k	move newer
n	next match
o	preview
p	diff current buffer
q	quit
r	diff
gg	move top
?	toggle help

Multi-Encodings

SpaceVim uses utf-8 as the default encoding. There are four options for this:

- fileencodings (fencs): ucs-bom,utf-8,default,latin1
- fileencoding (fenc): utf-8
- encoding (enc): utf-8
- termencoding (tenc): utf-8 (only supported in Vim)

To fix a messy display: `SPC e a` is the mapping to auto detect the file encoding. After detecting the file encoding, you can run the command below to fix it:

```
set enc=utf-8
write
```

Windows and Tabs

Windows Manager

Window manager key bindings can only be used in normal mode. The default leader `[WIN]` is `s`, you can change it via `windows_leader` in the `[options]` section:

```
[options]
    windows_leader = "s"
```

Key Bindings	Descriptions
`q`	Smart buffer close
`WIN v`	:split
`WIN V`	Split with previous buffer
`WIN g`	:vsplit
`WIN G`	Vertically split with previous buffer
`WIN t`	Open new tab (:tabnew)
`WIN o`	Close other windows (:only)
`WIN x`	Remove buffer, leave blank window
`WIN q`	Remove current buffer
`WIN Q`	Close current buffer (:close)
`Shift-Tab`	Switch to alternate window (switch back and forth)

SpaceVim has mapped normal `q` (record a macro) as smart buffer close, and record a macro (vim's `q`) has been mapped to `<Leader> q r`, if you want to disable this feature, you can use `vimcompatible` mode.

General Editor windows

Key Bindings	Descriptions
`<F2>`	Toggle tagbar
`<F3>`	Toggle Vimfiler
`Ctrl-Down`	Move to split below (`Ctrl-w j`)

Key Bindings	Descriptions
`Ctrl-Up`	Move to upper split (`Ctrl-w k`)
`Ctrl-Left`	Move to left split (`Ctrl-w h`)
`Ctrl-Right`	Move to right split (`Ctrl-w l`)

Window manipulation key bindings

Every window has a number displayed at the start of the statusline and can be quickly accessed using `SPC number`.

Key Bindings	Descriptions
`SPC 1`	go to window number 1
`SPC 2`	go to window number 2
`SPC 3`	go to window number 3
`SPC 4`	go to window number 4
`SPC 5`	go to window number 5
`SPC 6`	go to window number 6
`SPC 7`	go to window number 7
`SPC 8`	go to window number 8
`SPC 9`	go to window number 9

Windows manipulation commands (start with `w`):

Key Bindings	Descriptions
`SPC w .`	windows transient state
`SPC w <Tab>`	switch to alternate window in the current frame (switch back and forth)
`SPC w =`	balance split windows
`SPC w b`	force the focus back to the minibuffer (TODO)
`SPC w c`	Distraction-free reading current window (tools layer)
`SPC w C`	Distraction-free reading other windows via vim-choosewin (tools layer)
`SPC w d`	delete a window
`SPC u SPC w d`	delete a window and its current buffer (does not delete the file) (TODO)
`SPC w D`	delete another window using vim-choosewin
`SPC u SPC w D`	delete another window and its current buffer using vim-choosewin (TODO)
`SPC w t`	toggle window dedication (dedicated window cannot be reused by a mode) (TODO)

Key Bindings	Descriptions
SPC w f	toggle follow mode
SPC w F	create new tab
SPC w h	move to window on the left
SPC w H	move window to the left
SPC w j	move to window below
SPC w J	move window to the bottom
SPC w k	move to window above
SPC w K	move window to the top
SPC w l	move to window on the right
SPC w L	move window to the right
SPC w m	maximize/minimize a window
SPC w M	swap windows using vim-choosewin
SPC w o	cycle and focus between tabs
SPC w p m	open messages buffer in a popup window (TODO)
SPC w p p	close the current sticky popup window (TODO)
SPC w r	rotate windows forward
SPC w R	rotate windows backward
SPC w s / SPC w -	horizontal split
SPC w S	horizontal split and focus new window
SPC w u	undo window layout
SPC w U	redo window layout
SPC w v / SPC w /	vertical split
SPC w V	vertical split and focus new window
SPC w w	cycle and focus between windows
SPC w W	select window using vim-choosewin
SPC w x	exchange current window with next one

Tabs manipulation key bindings

Tab manipulation commands (start with F):

Key Bindings	Descriptions
`SPC F d`	close current tab
`SPC F D`	close other tabs
`SPC F n`	create a new tab

Buffers and Files

Buffers manipulation key bindings

Buffer manipulation commands (start with `b`):

Key Bindings	Descriptions
`SPC <Tab>`	switch to alternate buffer in the current window (switch back and forth)
`SPC b .`	buffer transient state
`SPC b b`	switch to a buffer (via denite/unite)
`SPC b d`	kill the current buffer (does not delete the visited file)
`SPC b D`	kill a visible buffer using vim-choosewin
`SPC b Ctrl-d`	kill other buffers
`SPC b Ctrl-Shift-d`	kill buffers using a regular expression
`SPC b e`	erase the content of the buffer (ask for confirmation)
`SPC b n`	switch to next buffer avoiding special buffers
`SPC b m`	open *Messages* buffer
`SPC b o`	kill all saved buffers and windows except the current one
`SPC b p`	switch to previous buffer avoiding special buffers
`SPC b P`	copy clipboard and replace buffer (useful when pasting from a browser)
`SPC b R`	revert the current buffer (reload from disk)
`SPC b s`	switch to the *scratch* buffer (create it if needed)
`SPC b w`	toggle read-only (writable state)
`SPC b Y`	copy whole buffer to clipboard (useful when copying to a browser)

Create a new empty buffer

Key Bindings	Descriptions
`SPC b N h`	create new empty buffer in a new window on the left

Key Bindings	Descriptions
SPC b N j	create new empty buffer in a new window at the bottom
SPC b N k	create new empty buffer in a new window above
SPC b N l	create new empty buffer in a new window below
SPC b N n	create new empty buffer in current window

Special Buffers

Buffers created by plugins are not normal files, and they will not be listed on tabline. And also will not be listed by SPC b b key binding in fuzzy finder layer.

File manipulation key bindings

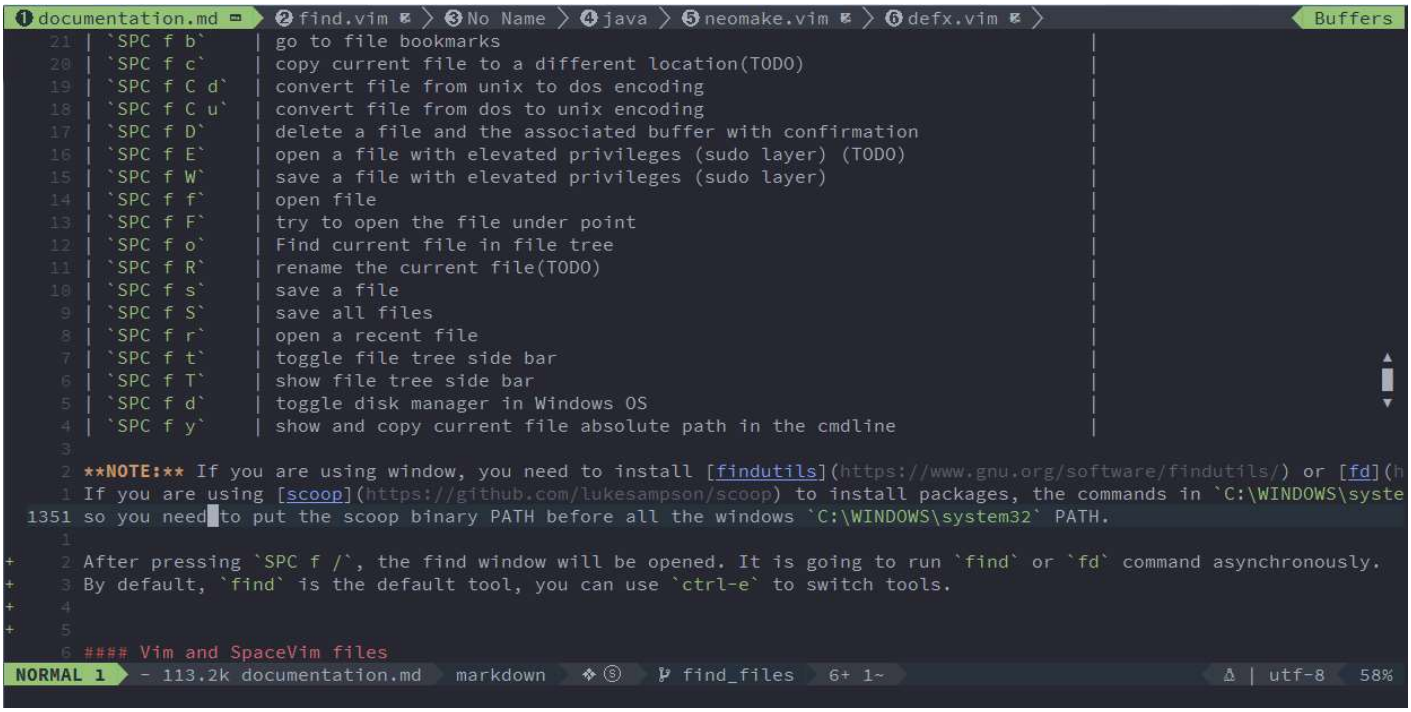
Files manipulation commands (start with f):

Key Bindings	Descriptions
SPC f /	Find files with find or fd command
SPC f b	go to file bookmarks
SPC f c	copy current file to a different location(TODO)
SPC f C d	convert file from unix to dos encoding
SPC f C u	convert file from dos to unix encoding
SPC f D	delete a file and the associated buffer with confirmation
SPC f E	open a file with elevated privileges (sudo layer) (TODO)
SPC f W	save a file with elevated privileges (sudo layer)
SPC f f	fuzzy find files in buffer directory
SPC f F	fuzzy find cursor file in buffer directory
SPC f o	Find current file in file tree
SPC f R	rename the current file
SPC f s	save a file
SPC f a	save as new file name
SPC f S	save all files
SPC f r	open a recent file
SPC f t	toggle file tree side bar
SPC f T	show file tree side bar

Key Bindings	Descriptions
SPC f d	toggle disk manager in Windows OS
SPC f y	show and copy current file absolute path in the cmdline
SPC f Y	show and copy remote url of current file

NOTE: If you are using Windows, you need to install findutils or fd. If you are using scoop to install packages, the commands in `C:\WINDOWS\system32` will override the User `PATH`, so you need to put the scoop binary path before `C:\WINDOWS\system32` in `PATH`.

After pressing `SPC f /`, the find window will be opened. It is going to run `find` or `fd` command asynchronously. By default, `find` is the default tool, you can use `ctrl-e` to switch tools.



To change the default file searching tool, you can use `file_searching_tools` in the `[options]` section. It is `[]` by default.

```
[options]
    file_searching_tools = ['find', 'find -not -iwholename "*.git*" ']
```

The first item is the name of the tool, the second one is the default searching command.

Vim and SpaceVim files

Convenient key bindings are located under the prefix `SPC f v` to quickly navigate between Vim and SpaceVim specific files.

Key Bindings	Descriptions
SPC f v v	display and copy SpaceVim version
SPC f v d	open SpaceVim custom configuration file

Key Bindings	Descriptions
`SPC f v s`	list all loaded vim scripts, like `:scriptnames`

Available layers

All layers can be easily discovered via `:SPLayer -l` accessible with `SPC h l`.

Available plugins in SpaceVim

All plugins can be easily discovered via `<Leader> f p`.

Fuzzy finder

Fuzzy finder provides a variety of efficient content searching key bindings, including file searching, outline searching, vim messages searching and register content searching.

Currently, there are six fuzzy finder layers:

- `unite` layer: based on `Shougo/unite.vim`
- `denite` layer: based on `Shougo/denite.nvim`
- `leaderf` layer: based on `Yggdroot/LeaderF`
- `ctrlp` layer: based on `ctrlpvim/ctrlp.vim`
- `fzf` layer: based on fzf
- `telescope` layer: based on telescope.nvim

These layers have the same key bindings and features. But they need different dependencies.

Users only need to load one of these layers to be able to get these features.

for example, to load the denite layer:

```
[[layers]]
    name = "denite"
```

Key bindings

Key bindings	Discription
`<Leader> f <Space>`	Fuzzy find menu:CustomKeyMaps
`<Leader> f p`	Fuzzy find menu:AddedPlugins
`<Leader> f e`	Fuzzy find register
`<Leader> f h`	Fuzzy find history/yank
`<Leader> f j`	Fuzzy find jump, change
`<Leader> f l`	Fuzzy find location list
`<Leader> f m`	Fuzzy find output messages
`<Leader> f o`	Fuzzy find outline
`<Leader> f q`	Fuzzy find quick fix

Key bindings	Discription
`<Leader> f r`	Resumes Unite window

Differences between these layers

The above key bindings are only part of fuzzy finder layers, please read the layers's documentations.

Feature	denite	unite	leaderf	ctrlp	fzf
CustomKeyMaps menu	yes	yes	yes	no	no
AddedPlugins menu	yes	yes	yes	no	no
register	yes	yes	yes	yes	yes
file	yes	yes	yes	yes	yes
yank history	yes	yes	yes	no	yes
jump	yes	yes	yes	yes	yes
location list	yes	yes	yes	no	yes
outline	yes	yes	yes	yes	yes
message	yes	yes	yes	no	yes
quickfix list	yes	yes	yes	yes	yes
resume windows	yes	yes	yes	no	no

Key bindings within the fuzzy finder buffer

Key Bindings	Descriptions
`<Tab>` / `Ctrl-j`	Select next line
`Shift-Tab` / `Ctrl-k`	Select previous line
`<Esc>`	Leave Insert mode
`Ctrl-w`	Delete backward path
`Ctrl-u`	Delete whole line before cursor
`<Enter>`	Run default action
`Ctrl-s`	Open in a split
`Ctrl-v`	Open in a vertical split
`Ctrl-t`	Open in a new tab
`Ctrl-g`	Close fuzzy finder

With an external tool

SpaceVim can be interfaced with different searching tools like:

- rg - ripgrep
- ag - the silver searcher
- pt - the platinum searcher
- ack
- grep

The search commands in SpaceVim are organized under the `SPC s` prefix with the next key being the tool to use and the last key is the scope. For instance, `SPC s a b` will search in all opened buffers using `ag`.

If the last key (determining the scope) is uppercase then the current word under the cursor is used as default input for the search. For instance, `SPC s a B` will search for the word under the cursor.

If the tool key is omitted then a default tool will be automatically selected for the search. This tool corresponds to the first tool found on the system from the list `search_tools`, the default order is `['rg', 'ag', 'pt', 'ack', 'grep', 'findstr', 'git']`. For instance `SPC s b` will search in the opened buffers using `pt` if `rg` and `ag` have not been found on the system.

The tool keys are:

Tool	Key
ag	a
grep	g
git grep	G
ack	k
rg	r
pt	t

The available scopes and corresponding keys are:

Scope	Key
opened buffers	b
buffer directory	d
files in a given directory	f
current project	p

Notes:

- `rg`, `ag` and `pt` are optimized to be used in a source control repository but they can be used in an arbitrary directory as well.
- It is also possible to search in several directories at once by marking them in the unite buffer.

Beware if you use `pt`, TCL parser tools also install a command line tool called `pt`.

Custom searching tool

To change the options of a search tool, you need to use the bootstrap function. The following example shows how to change the default option of searching tool `rg`.

```
function! myspacevim#before() abort
    let profile = SpaceVim#mapping#search#getprofile('rg')
    let default_opt = profile.default_opts + ['--no-ignore-vcs']
    call SpaceVim#mapping#search#profile({'rg' : {'default_opts' : default_opt}})
endfunction
```

The structure of searching tool profile is:

```
" { 'ag' : {
"   'namespace' : '',         " a single char a-z
"   'command' : '',           " executable
"   'default_opts' : [],      " default options
"   'recursive_opt' : [],     " default recursive options
"   'expr_opt' : '',          " option to enable expr mode
"   'fixed_string_opt' : '',  " option to enable fixed string mode
"   'ignore_case' : '',       " option to enable ignore case mode
"   'smart_case' : '',        " option to enable smart case mode
"   }
"  }
```

Useful key bindings

Key Bindings	Descriptions
`SPC r l`	resume the last completion buffer
`SPC s ` `	go back to the previous place before jump
Prefix argument	will ask for file extensions

Summary

The following table summurizes the search key bindings:

Key Bindings	Descriptions
`SPC s <scope>`	Search using the first found tool
`SPC s a <scope>`	Search using `ag`
`SPC s g <scope>`	Search using `grep`
`SPC s G <scope>`	Search using `git-grep`
`SPC s k <scope>`	Search using `ack`

Key Bindings	Descriptions
`SPC s r <scope>`	Search using `rg`
`SPC s t <scope>`	Search using `pt`
`SPC s /`	Search in the project on the fly using the default tools
`SPC s w g`	Search google (opens search results in a browser) (TODO)
`SPC s w w`	Search wikipedia (opens search results in a browser) (TODO)

With `<scope>` being one of the following:

Scope	Description
`b`	All open buffers
`d`	Current buffer's directory
`f`	Arbitrary directory
`p`	Current project
`s`	Current buffer
`j`	Background search in the project

Notes:

- A capital letter may be used for `<scope>` to search for the word under the cursor.
- To enable google suggestions, you need to add `enable_googlesuggest = 1` to your custom configurations file.

Hint: It is also possible to search in a project without having to open a file beforehand. To do so use `[SPC] p p` and then `C-s` on a given project to directly search into it like with `[SPC] s p`. (TODO)

Key bindings in FlyGrep buffer:

Key Bindings	Descriptions
`<Esc>`	close FlyGrep buffer
`<Enter>`	open file at the cursor line
`Ctrl-t`	open item in new tab
`Ctrl-s`	open item in split window
`Ctrl-v`	open item in vertical split window
`Ctrl-q`	apply all items into quickfix
`<Tab>`	move cursor line down
`Shift-<Tab>`	move cursor line up
`<BackSpace>`	remove last character
`Ctrl-w`	remove the Word before the cursor

Key Bindings	Descriptions
`Ctrl-u`	remove the Line before the cursor
`Ctrl-k`	remove the Line after the cursor
`Ctrl-a` / `<Home>`	Go to the beginning of the line
`Ctrl-e` / `<End>`	Go to the end of the line

Persistent highlighting

SpaceVim uses `search_highlight_persist` to keep the searched expression highlighted until the next search. It is also possible to clear the highlighting by pressing `[SPC] s c` or executing the ex command `:noh`.

Getting help

Fuzzy finder layer is powerful tool to unite all interfaces. It is meant to be like Helm for Vim. These mappings are for getting help info about functions, variables etc:

Key Bindings	Descriptions
`SPC h SPC`	discover SpaceVim documentation, layers and packages using fuzzy finder layer
`SPC h i`	get help with the symbol at point
`SPC h g`	run `:helpgrep` asynchronously
`SPC h G`	run `:helpgrep` asynchronously with the word under cursor
`SPC h k`	show top-level bindings with which-key
`SPC h m`	search available man pages

NOTE: `SPC h i` and `SPC h m` need to load at least one fuzzy finder layer.

Reporting an issue:

Key Bindings	Descriptions
`SPC h I`	Open SpaceVim GitHub issue template with pre-filled information

Unimpaired bindings

Mappings	Descriptions
`[ SPC`	Insert space above
`] SPC`	Insert space below
`[ b`	Go to previous buffer
`] b`	Go to next buffer

Mappings	Descriptions
`[ n`	Go to previous conflict marker
`] n`	Go to next conflict marker
`[ f`	Go to previous file in directory
`] f`	Go to next file in directory
`[ l`	Go to the previous error
`] l`	Go to the next error
`[ c`	Go to the previous vcs hunk (need VersionControl layer)
`] c`	Go to the next vcs hunk (need VersionControl layer)
`[ q`	Go to the previous error
`] q`	Go to the next error
`[ t`	Go to the previous frame
`] t`	Go to the next frame
`[ w`	Go to the previous window
`] w`	Go to the next window
`[ e`	Move line up
`] e`	Move line down
`[ p`	Paste above current line
`] p`	Paste below current line
`g p`	Select pasted text

Jumping, Joining and Splitting

The `SPC j` prefix is for jumping, joining and splitting.

Jumping

Key Bindings	Descriptions
`SPC j $`	go to the end of line (and set a mark at the previous location in the line)
`SPC j 0`	go to the beginning of line (and set a mark at the previous location in the line)
`SPC j b`	jump backward
`SPC j c`	jump to last change

Key Bindings	Descriptions
SPC j d	jump to a listing of the current directory
SPC j D	jump to a listing of the current directory (other window)
SPC j f	jump forward
SPC j I	jump to a definition in any buffer (denite outline)
SPC j i	jump to a definition in buffer (denite outline)
SPC j j	jump to a character in the buffer (easymotion)
SPC j J	jump to a suite of two characters in the buffer (easymotion)
SPC j k	jump to next line and indent it using auto-indent rules
SPC j l	jump to a line with avy (easymotion)
SPC j q	show the dumb-jump quick look tooltip (TODO)
SPC j u	jump to a URL in the current window
SPC j v	jump to the definition/declaration of an Emacs Lisp variable (TODO)
SPC j w	jump to a word in the current buffer (easymotion)

Joining and splitting

Key Bindings	Descriptions
J	join the current line with the next line
SPC j o	join a code block into a single-line statement
SPC j m	split a one-liner into multiple lines
SPC j k	go to next line and indent it using auto-indent rules
SPC j n	split the current line at point, insert a new line and auto-indent
SPC j o	split the current line at point but let point on current line
SPC j s	split a quoted string or s-expression in place
SPC j S	split a quoted string or s-expression with \n, and auto-indent the new line

Other key bindings

Commands starting with g

After pressing prefix g in normal mode, if you do not remember the mappings, you will see the guide which contains short descriptions of all the mappings starting with g.

Key Bindings	Descriptions
g #	search under cursor backward
g $	go to rightmost character
g &	repeat last ":s" on all lines
g '	jump to mark
g *	search under cursor forward
g +	newer text state
g ,	newer position in change list
g -	older text state
g /	stay incsearch
g 0	go to leftmost character
g ;	older position in change list
g <	last page of previous command output
g <Home>	go to leftmost character
g E	end of previous word
g F	edit file under cursor(jump to line after name)
g H	select line mode
g I	insert text in column 1
g J	join lines without space
g N	visually select previous match
g Q	switch to Ex mode
g R	enter VREPLACE mode
g T	previous tag page
g U	make motion text uppercase
g]	tselect cursor tag
g ^	go to leftmost no-white character
g _	go to last char
g `	jump to mark
g a	print ascii value of cursor character
g d	goto definition

Key Bindings	Descriptions
`g e`	go to end of previous word
`g f`	edit file under cursor
`g g`	go to line N
`g h`	select mode
`g i`	insert text after ‘^ mark
`g j`	move cursor down screen line
`g k`	move cursor up screen line
`g m`	go to middle of screenline
`g n`	visually select next match
`g o`	goto byte N in the buffer
`g p`	Select last paste
`g s`	sleep N seconds
`g t`	next tag page
`g u`	make motion text lowercase
`g ~`	swap case for Nmove text
`g <End>`	go to rightmost character
`g Ctrl-g`	show cursor info

Commands starting with `z`

After pressing prefix `z` in normal mode, if you do not remember the mappings, you will see the guide which contains short descriptions of all the mappings starting with `z`.

Key Bindings	Descriptions
`z <Right>`	scroll screen N characters to left
`z +`	cursor to screen top line N
`z -`	cursor to screen bottom line N
`z .`	cursor line to center
`z <Enter>`	cursor line to top
`z =`	spelling suggestions
`z A`	toggle folds recursively
`z C`	close folds recursively

Key Bindings	Descriptions
`z D`	delete folds recursively
`z E`	eliminate all folds
`z F`	create a fold for N lines
`z G`	mark good spelled (update internal wordlist)
`z H`	scroll half a screenwidth to the right
`z L`	scroll half a screenwidth to the left
`z M`	set `foldlevel` to zero
`z N`	set `foldenable`
`z O`	open folds recursively
`z R`	set `foldlevel` to deepest fold
`z W`	mark wrong spelled (update internal wordlist)
`z X`	re-apply `foldlevel`
`z ^`	cursor to screen bottom line N
`z a`	toggle a fold
`z b`	redraw, cursor line at bottom
`z c`	close a fold
`z d`	delete a fold
`z e`	right scroll horizontally to cursor position
`z f`	create a fold for motion
`z g`	mark good spelled
`z h`	scroll screen N characters to right
`z i`	toggle foldenable
`z j`	mode to start of next fold
`z k`	mode to end of previous fold
`z l`	scroll screen N characters to left
`z m`	subtract one from `foldlevel`
`z n`	reset `foldenable`
`z o`	open fold
`z r`	add one to `foldlevel`

Key Bindings	Descriptions
`z s`	left scroll horizontally to cursor position
`z t`	cursor line at top of window
`z v`	open enough folds to view cursor line
`z w`	mark wrong spelled
`z x`	re-apply foldlevel and do “zV”
`z z`	smart scroll
`z <Left>`	scroll screen N characters to right

Advanced usage

Managing projects

When you open a file, SpaceVim will change the current directory to the root directory of the project that contains this file. The project's root directory detection is based on the `project_rooter_patterns` in the `[options]` section, and the default value is:

```
[options]
    project_rooter_patterns = ['.git/', '_darcs/', '.hg/', '.bzr/', '.svn/']
```

The project manager will find the outermost directory by default. To find the nearest directory instead, you need to change `project_rooter_outermost` to `false`:

```
[options]
    project_rooter_patterns = ['.git/', '_darcs/', '.hg/', '.bzr/', '.svn/']
    project_rooter_outermost = false
```

Sometimes we want to ignore some directories when detecting the project root directory. To do that add a `!` prefix before the pattern. For example, to ignore the `node_packages/` directory:

```
[options]
    project_rooter_patterns = ['.git/', '_darcs/', '.hg/', '.bzr/', '.svn/', '!no
    project_rooter_outermost = false
```

There are three options for non-project files/directories:

- Don't change directory (default).

```
project_non_root = ''
```

- Change to file's directory (similar to 'autochdir').

```
project_non_root = 'current'
```

- Change to home directory.

```
project_non_root = 'home'
```

You can also disable project root detection completely (i.e. vim will set the root directory to the present working directory):

```
[options]
    project_auto_root = false
```

Project manager commands start with `p`:

Key Bindings	Descriptions
`SPC p '`	open a shell in project's root (need the shell layer)

Show project info on cmdline

By default the key binding `Ctrl-g` will display the information of current project on command line.

Searching files in project

Key Bindings	Descriptions
`SPC p f` / `Ctrl-p`	find files in current project
`SPC p F`	find cursor file in current project
`SPC p /`	fuzzy search for text in current project
`SPC p k`	kill all buffers of current project
`SPC p p`	list all projects

`SPC p p` will list all the projects history cross vim sessions. By default only 20 projects will be listed. To increase it, you can change the value of `projects_cache_num`.

To disable the cross session cache, change `enable_projects_cache` to `false`.

```
[options]
    enable_projects_cache = true
    projects_cache_num = 20
```

Custom alternate file

To manage the alternate file of the project, you need to create a `.project_alt.json` file in the root of your project. Then you can use the command `:A` to jump to the alternate file of current file. You can also specific the type of alternate file, for example `:A doc`. With a bang `:A!`, SpaceVim will parse the configuration file additionally. If no type is specified, the default type `alternate` will be used.

here is an example of `.project_alt.json`:

```
{
  "autoload/SpaceVim/layers/lang/*.vim": {
    "doc": "docs/layers/lang/{}.md",
    "test": "test/layer/lang/{}.vader"
  }
}
```

Instead of using json file, the alternate file manager also support toml file, for example:

```
["autoload/SpaceVim/layers/lang/*.vim"]
    # You can use comments in toml file.
    doc = "docs/layers/lang/{}.md"
    test = "test/layer/lang/{}.vader"
```

If you do not want to use configuration file, or want to override the default configuration in alternate config file, `b:alternate_file_config` can be used in bootstrap function, for example:

```
augroup myspacevim
    autocmd!
    autocmd BufNewFile,BufEnter *.c let b:alternate_file_config = {
        \ "src/*.c" : {
            \ "doc" : "docs/{}.md",
            \ "alternate" : "include/{}.h",
            \ }
        \ }
    autocmd BufNewFile,BufEnter *.h let b:alternate_file_config = {
        \ "include/*.h" : {
            \ "alternate" : "scr/{}.c",
            \ }
        \ }
augroup END
```

Bookmarks management

Bookmarks manager is included in `tools` layer, to use the following key bindings, you need to enable the `tools` layer:

```
[[layers]]
    name = "tools"
```

Key Bindings	Descriptions
`m a`	Show list of all bookmarks
`m c`	Removes bookmarks for current buffer
`m m`	Toggle bookmark in current line

Key Bindings	Descriptions
`m n`	Jump to next bookmark
`m p`	Jump to previous bookmark
`m i`	Annotate bookmark

As SpaceVim uses the above mappings, you cannot use the `a`, `c`, `m`, `n`, `p` or `i` registers to mark the current position, but other registers should work well. If you really need to use these registers, you can map `<Leader> m` to `m` in your bootstrap function, then you can use the registers via `<Leader> m <register>`.

```
function! myspacevim#before() abort
    nnoremap <silent><Leader>m m
endfunction
```

Tasks

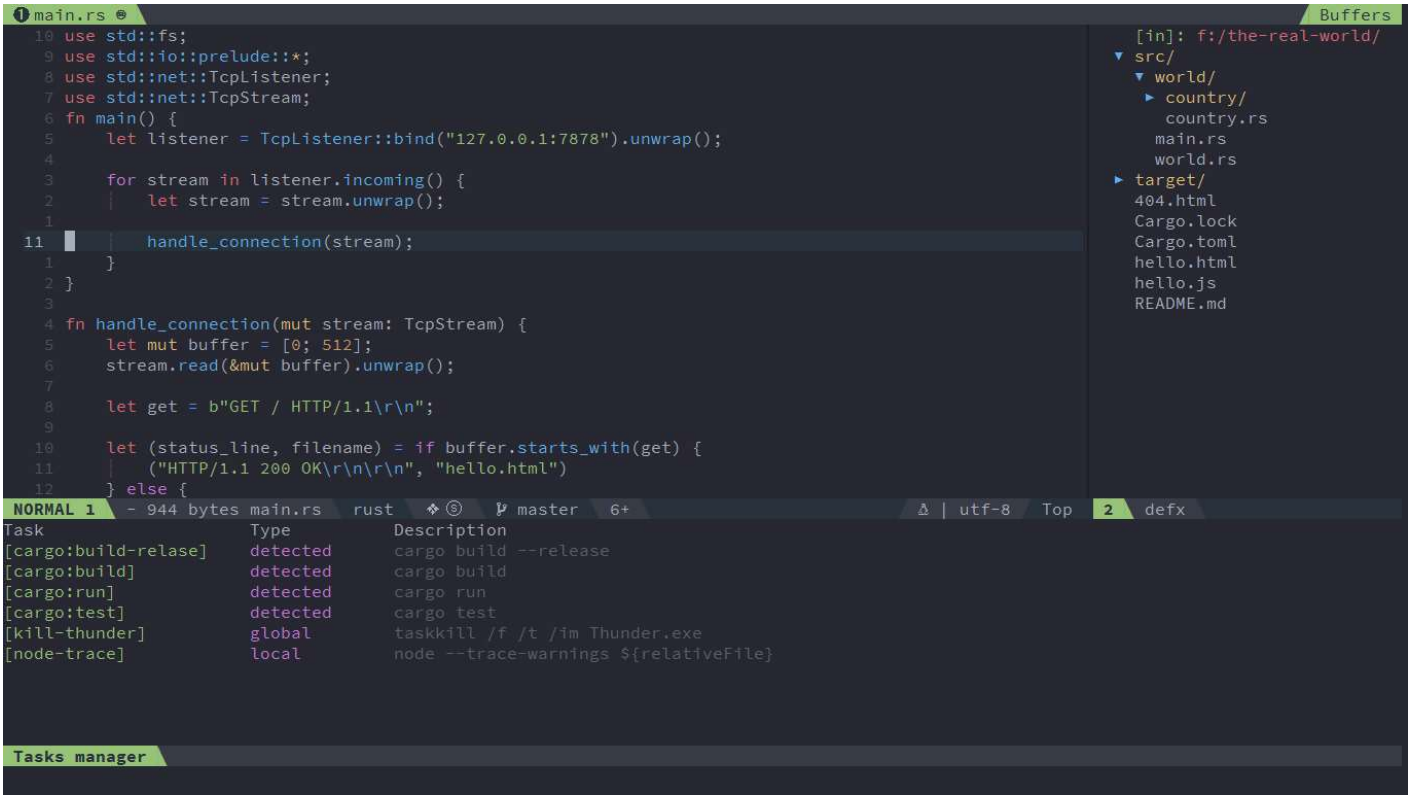
To integrate with external tools, SpaceVim introduced a task manager system, which is similar to VSCode's tasks-manager. There are two kinds of task configurations file:

- `~/.SpaceVim.d/tasks.toml`: global tasks configuration
- `.SpaceVim.d/tasks.toml`: project local tasks configuration

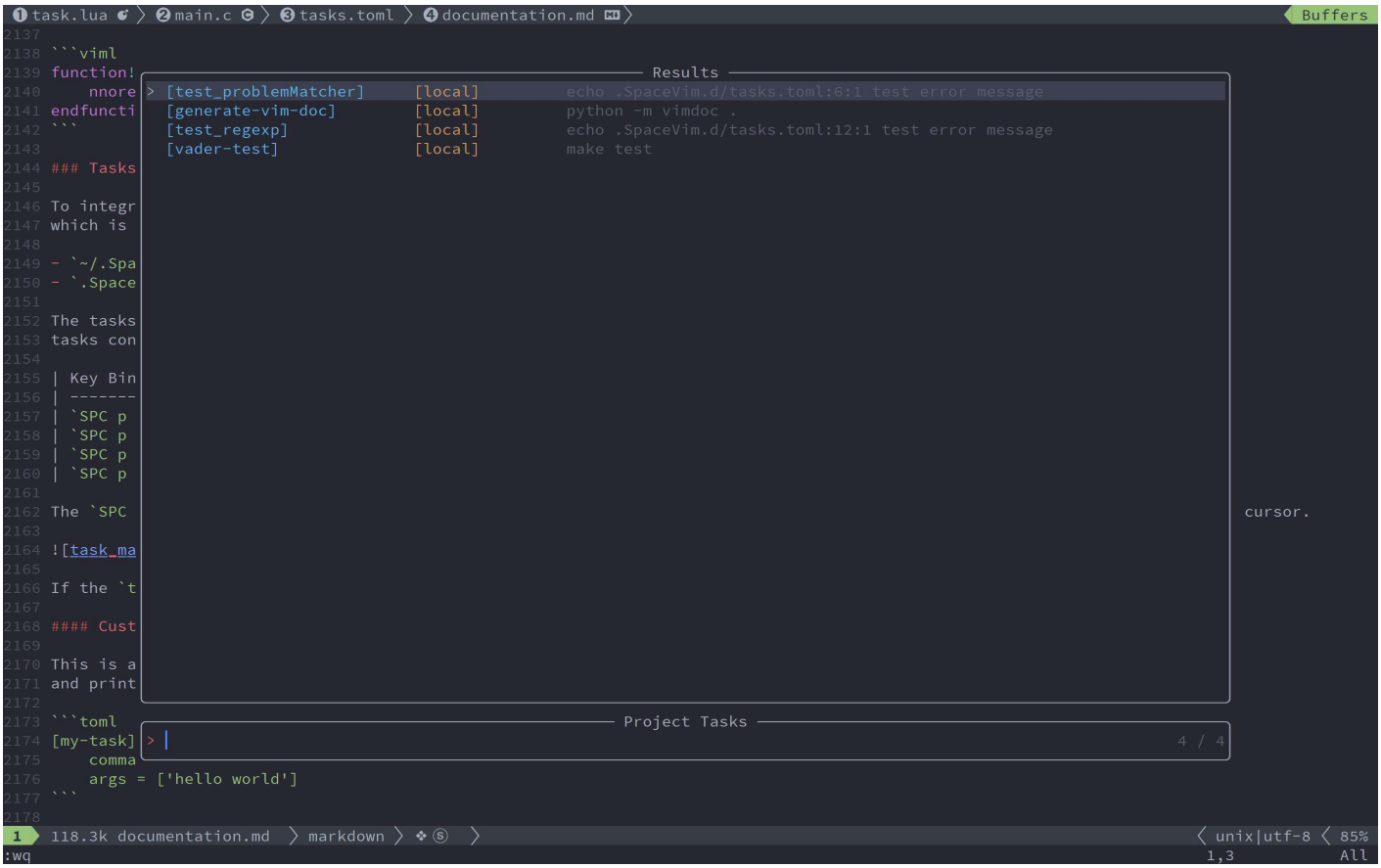
The tasks defined in the global tasks configuration can be overrided by project local tasks configuration.

Key Bindings	Descriptions
`SPC p t e`	edit tasks configuration file
`SPC p t r`	select task to run
`SPC p t l`	list all available tasks
`SPC p t f`	fuzzy find tasks(require telescope layer)

The `SPC p t l` will open the tasks manager windows, in the tasks manager windows, you can use `Enter` to run task under the cursor.



If the `telescope` layer is loaded, you can also use `SPC p t f` to fuzzy find specific task, and run the select task.



Custom tasks

This is a basic task configuration for running `echo hello world`, and print the results to the runner window.

```
[my-task]
    command = 'echo'
    args = ['hello world']
```

To run the task in the background, you need to set `isBackground` to `true`:

```
[my-task]
    command = 'echo'
    args = ['hello world']
    isBackground = true
```

The following task properties are available:

Name	Description
command	The actual command to execute.
args	The arguments passed to the command, it should be a list of strings and may be omitted.
options	Override the defaults for `cwd`,`env` or `shell`.
isBackground	Specifies whether the task should run in the background. by default, it is `false`.
description	Short description of the task
problemMatcher	Problems matcher of the task

Note: When a new task is executed, it will kill the previous task. If you want to keep the task, run it in background by setting `isBackground` to `true`.

SpaceVim supports variable substitution in the task properties, The following predefined variables are supported:

Name	Description
${workspaceFolder}	The project's root directory
${workspaceFolderBasename}	The name of current project's root directory
${file}	The path of current file
${relativeFile}	The current file relative to project root

Name	Description
${relativeFileDirname}	The current file's dirname relative to workspaceFolder
${fileBasename}	The current file's basename
${fileBasenameNoExtension}	The current file's basename without file extension
${fileDirname}	The current file's dirname
${fileExtname}	The current file's extension
${cwd}	The task runner's current working directory on startup
${lineNumber}	The current selected line number in the active file

For example: Supposing that you have the following requirements:

A file located at `/home/your-username/your-project/folder/file.ext` opened in your editor; The directory `/home/your-username/your-project` opened as your root workspace. So you will have the following values for each variable:

Name	Value
${workspaceFolder}	`/home/your-username/your-project/`
${workspaceFolderBasename}	`your-project`
${file}	`/home/your-username/your-project/folder/file.ext`
${relativeFile}	`folder/file.ext`
${relativeFileDirname}	`folder/`
${fileBasename}	`file.ext`
${fileBasenameNoExtension}	`file`
${fileDirname}	`/home/your-username/your-project/folder/`
${fileExtname}	`.ext`
${lineNumber}	line number of the cursor

Task Problems Matcher

Problem matcher is used to capture the message in the task output and show a corresponding problem in quickfix windows.

`problemMatcher` supports `errorformat` and `pattern` properties.

If the `errorformat` property is not defined, the `&errorformat` option will be used.

```
[test_problemMatcher]
    command = "echo"
    args = ['.SpaceVim.d/tasks.toml:6:1 test error message']
    isBackground = true
```

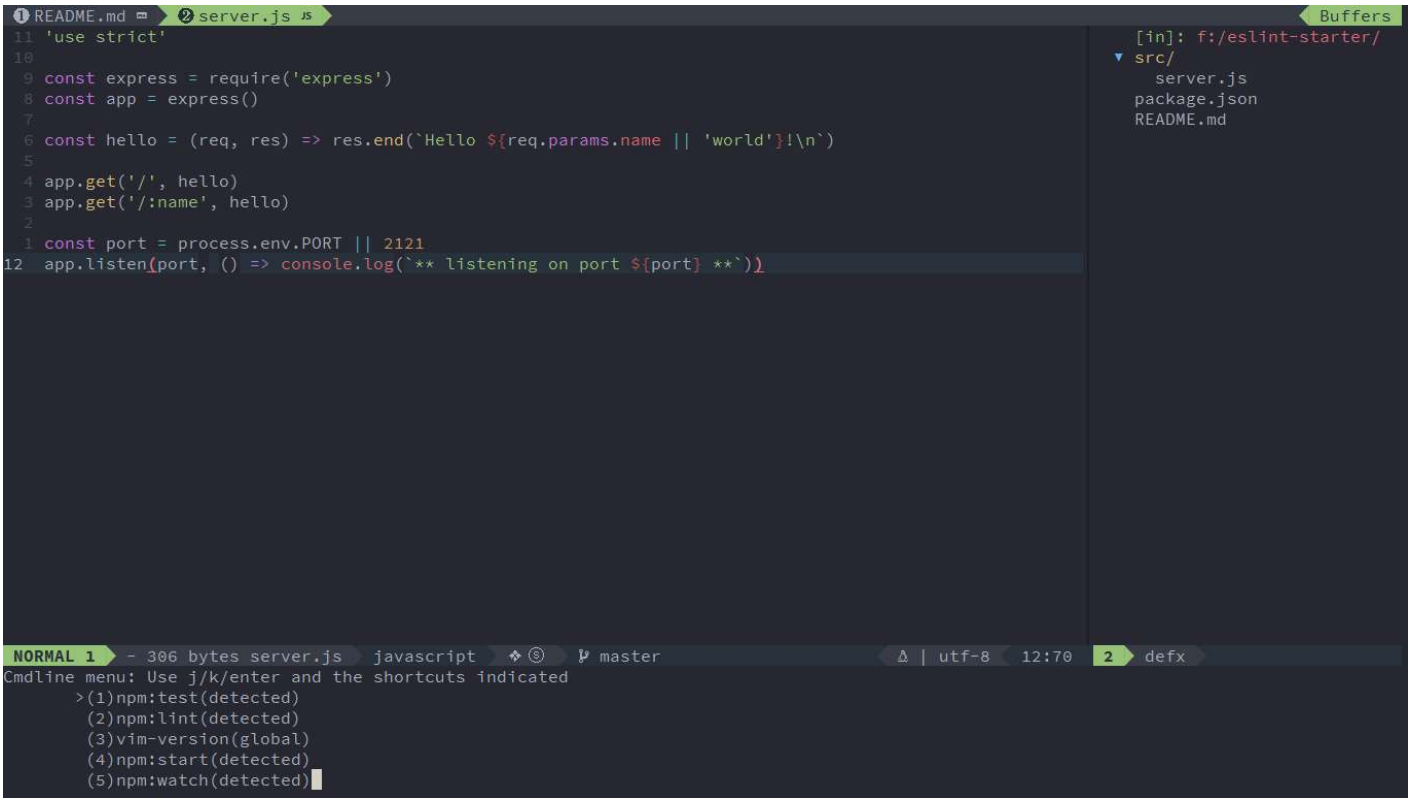
```
[test_problemMatcher.problemMatcher]
    useStdout = true
    errorformat = '%f:%l:%c\ %m'
```

If `pattern` is defined, the `errorformat` option will be ignored. Here is an example:

```
[test_regexp]
    command = "echo"
    args = ['.SpaceVim.d/tasks.toml:12:1 test error message']
    isBackground = true
[test_regexp.problemMatcher]
    useStdout = true
[test_regexp.problemMatcher.pattern]
      regexp = '\(.*\):\(\d\+\):\(\d\+\)\s\(\s.*\)'
      file = 1
      line = 2
      column = 3
      #severity = 4
      message = 4
```

Task auto-detection

Currently, SpaceVim can auto-detect tasks for npm. the tasks manager will parse the `package.json` file for npm packages. If you have cloned the eslint-starter. for example, pressing `SPC p t r` shows the following list:



Task provider

Some tasks can be automatically detected by the task provider. For example, a Task Provider could check if there is a specific build file, such as `package.json`, and create npm tasks.

To build a task provider, you need to use the Bootstrap function. The task provider should be a vim function that returns a task object.

here is an example for building a task provider.

```
function! s:make_tasks() abort
    if filereadable('Makefile')
        let subcmds = filter(readfile('Makefile', ''), "v:val=~#'^.PHONY'")
        let conf = {}
        for subcmd in subcmds
            let commands = split(subcmd)[1:]
            for cmd in commands
                call extend(conf, {
                            \ cmd : {
                            \ 'command': 'make',
                            \ 'args' : [cmd],
                            \ 'isDetected' : 1,
                            \ 'detectedName' : 'make:'
                            \ }
                            \ })
            endfor
        endfor
        return conf
    else
        return {}
    endif
endfunction
call SpaceVim#plugins#tasks#reg_provider(function('s:make_tasks'))
```

The provider also can be implemented in lua, for example:

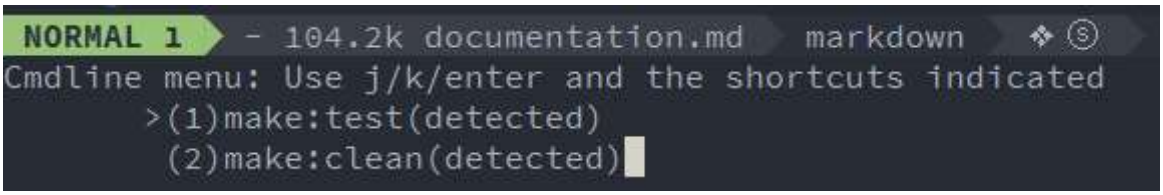
```
local task = require('spacevim.plugin.tasks')

local function make_tasks()
  if vim.fn.filereadable('Makefile') then
    local subcmds = {}
    local conf = {}
    for _, v in ipairs(vim.fn.readfile('Makefile', '')) do
      if vim.startwith(v, '.PHONY') then
        table.insert(subcmds, v)
      end
    end
    for _, subcmd in ipairs(subcmds) do
      local comamnds = vim.fn.split(subcmd)
```

```
      table.remove(commands, 1)
      for _, cmd in ipairs(commands) do
        conf = vim.tbl_extend('forces', conf, {
          [cmd] = {
            command = 'make',
            args = {cmd}
            isDetected = true,
            detectedName = 'make:'
          }
        })
      end
    end
    return conf
  else
    return {}
  end
end

task.reg_provider(make_tasks)
```

With the above configuration, you will see the following tasks in the SpaceVim repo:

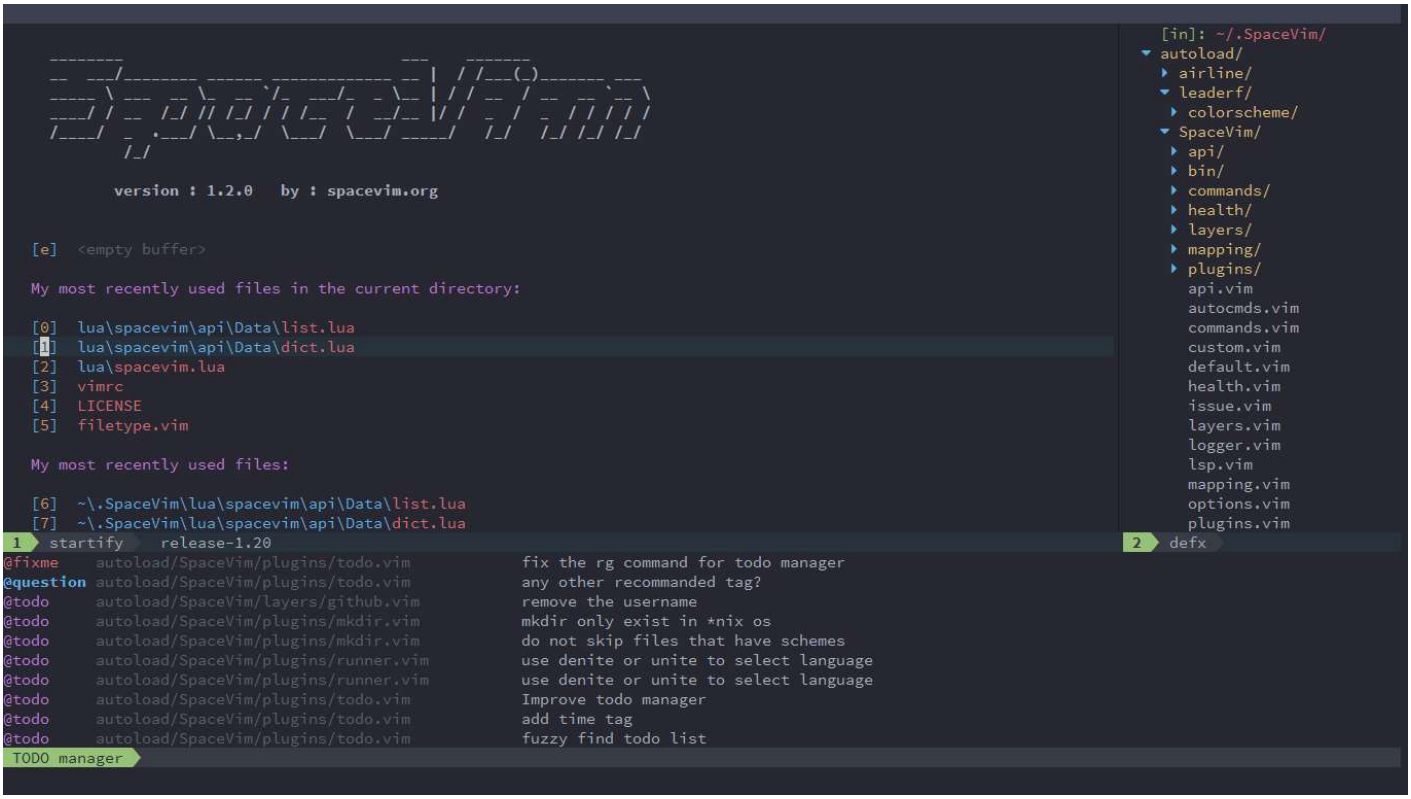


Todo manager

The todo manager plugin will run `rg` asynchronously, the results will be displayed on todo manager windows. The key binding is `SPC a o`. The default `todo_prefix` option is `@`, and the `todo_labels` is: `['fixme', 'question', 'todo', 'idea']`.

Example:

```
[options]
   todo_labels = ['fixme', 'question', 'todo', 'idea']
   todo_prefix = '@'
```



Known bug:

If you are using windows, and `grep.exe` do not support searching in subdirectory. and the stderr will shown:

```
[     todo ] [00:00:03:107] [ Debug ] stderr: grep.exe: ./wiki: warning: recursiv
```

To fix this issue, you need to install other searching tool, for example `rg`. and change the search_tools option:

```
[options]
    search_tools = ["rg", "ag", "grep"]
```

Replace text with iedit

SpaceVim uses a powerful iedit mode to quickly edit multiple occurrences of a symbol or selection.

Two new modes: `iedit-Normal`/`iedit-Insert`

The default color for iedit is `red`/`green` which is based on the current colorscheme.

iedit states key bindings

State transitions:

Key Bindings	Description
SPC s e	start iedit with all matchs
SPC s E	start iedit with only current match

In iedit-Normal mode:

`iedit-Normal` mode inherits from `Normal` mode, the following key bindings are specific to `iedit-Normal` mode.

Key Binding	Descriptions
`<Esc>`	go back to `Normal` mode
`i`	start `iedit-Insert` mode after current character
`a`	start `iedit-Insert` mode before current character
`I`	goto the beginning and start `iedit-Insert` mode
`A`	goto the end and start `iedit-Insert` mode
`<Left>`/`h`	Move cursor to left
`<Right>`/`l`	Move cursor to right
`0`/`<Home>`	go to the beginning of the current occurrence
`$`/`<End>`	go to the end of the current occurrence
`C`	delete from the cursor position to the end and start `iedit-Insert` mode
`D`	delete the occurrences
`s`	delete the character under cursor and start iedit-Insert mode
`S`	delete the occurrences and start iedit-Insert mode
`x`	delete the character under cursor in all the occurrences
`X`	delete the character before cursor in all the occurrences
`gg`	go to first occurrence
`G`	go to last occurrence
`f{char}`	To first occurrence of `{char}` to the right.
`n`	go to next occurrence
`N`	go to previous occurrence
`p`	replace occurrences with last yanked (copied) text
`<Tab>`	toggle current occurrence
`Ctrl-n`	forward and active next match
`Ctrl-x`	inactivate current match and move forward
`Ctrl-p`	inactivate current match and move backward
`e`	forward to the end of word
`w`	forward to the begin of next word
`b`	move to the begin of current word

In iedit-Insert mode:

Key Bindings	Descriptions
`Ctrl-g` / `<Esc>`	go back to `iedit-Normal` mode
`Ctrl-b` / `<Left>`	move cursor to left
`Ctrl-f` / `<Right>`	move cursor to right
`Ctrl-a` / `<Home>`	moves the cursor to the beginning of the current occurrence
`Ctrl-e` / `<End>`	moves the cursor to the end of the current occurrence
`Ctrl-w`	delete word before cursor
`Ctrl-k`	delete all words after cursor
`Ctrl-u`	delete all characters before cursor
`Ctrl-h` / `<Backspace>`	delete character before cursor
`<Delete>`	delete character after cursor

Code runner

SpaceVim provides an asynchronous code runner plugin. In most language layers, the key binding `SPC l r` is defined for running the current buffer. To close the code runner windows, you can use `Ctrl-`` key binding. If you need to add new commands, you can use the bootstrap function. For example: Use `F5` to build the project asynchronously.

```
nnoremap <silent> <F5> :call SpaceVim#plugins#runner#open('make')
```

Key bindings within code runner buffer:

key binding	description
`ctrl-c`	stop code runner
`i`	open promote to insert text

Custom runner

If you want to set custom code runner for specific language. You need to use `SpaceVim#plugins#runner#reg_runner(ft, runner)` api in bootstrap function.

example:

```
call SpaceVim#plugins#runner#reg_runner('lua', {
      \ 'exe' : 'lua',
      \ 'opt' : ['-'],
      \ 'usestdin' : 1,
      \ })
```

REPL(read eval print loop)

The REPL(Read Eval Print Loop) plugin provides a framework to run REPL command asynchronously.

For different language, you need to checkout the doc of language layer. The repl key bindings are defined in language layer.

Key bindings within repl buffer:

key binding	description
`i`	open promote to insert text

Highlight current symbol

SpaceVim supports highlighting current symbol on demand and add a transient state to easily navigate and rename these symbols.

It is also possible to change the range of the navigation on the fly, the available ranges are:

1. buffer: the whole buffer
2. function: in current function
3. visible area: in current visible area of the buffer

The default key binding to Highlight the symbol under the cursor is `SPC s h`.

Key Bindings	Descriptions
`*`	highlight current symbol and jump forwards
`#`	highlight current symbol and jump backwards
`SPC s e`	start iedit mode on current symbol
`SPC s h`	highlight current symbol within default range
`SPC s H`	highlight last symbol within default range

In highlight symbol transient state, the following key bindings can be used:

Key Bindings	Descriptions
`e`	start iedit mode
`n`	go to next occurrence
`N` / `p`	go to previous occurrence
`b`	search occurrence in all buffers
`/`	search occurrence in whole project
`<Tab>`	toggle highlight current occurrence
`r`	change range
`R`	go to home occurrence
Any other key	leave the navigation transient state

Error handling

SpaceVim uses neomake to give error feedback on the fly. The checks are only performed at save time by default.

Error management mappings (start with e):

Mappings	Descriptions
`SPC t s`	toggle syntax checker
`SPC e c`	clear all errors
`SPC e h`	describe a syntax checker
`SPC e l`	toggle the display of the list of errors/warnings
`SPC e n`	go to the next error
`SPC e p`	go to the previous error
`SPC e v`	verify syntax checker setup (useful to debug 3rd party tools configuration)
`SPC e .`	error transient state

The next/previous error mappings and the error transient state can be used to browse errors from syntax checkers as well as errors from location list buffers, and indeed anything that supports Vim's location list. This includes for example search results that have been saved to a location list buffer.

Custom sign symbol:

Symbol	Descriptions	Custom options
✕	Error	`error_symbol`
➤	warning	`warning_symbol`
ⓘ	Info	`info_symbol`

quickfix list navigation:

Mappings	Descriptions
`<Leader> q l`	Open quickfix list window
`<Leader> q c`	clear quickfix list
`<Leader> q n`	jump to next item in quickfix list
`<Leader> q p`	jump to previous item in quickfix list

EditorConfig

SpaceVim supports EditorConfig, a configuration file to "define and maintain consistent coding styles between different editors and IDEs."

To customize your editorconfig experience, read the editorconfig-vim package's documentation.

Vim Server

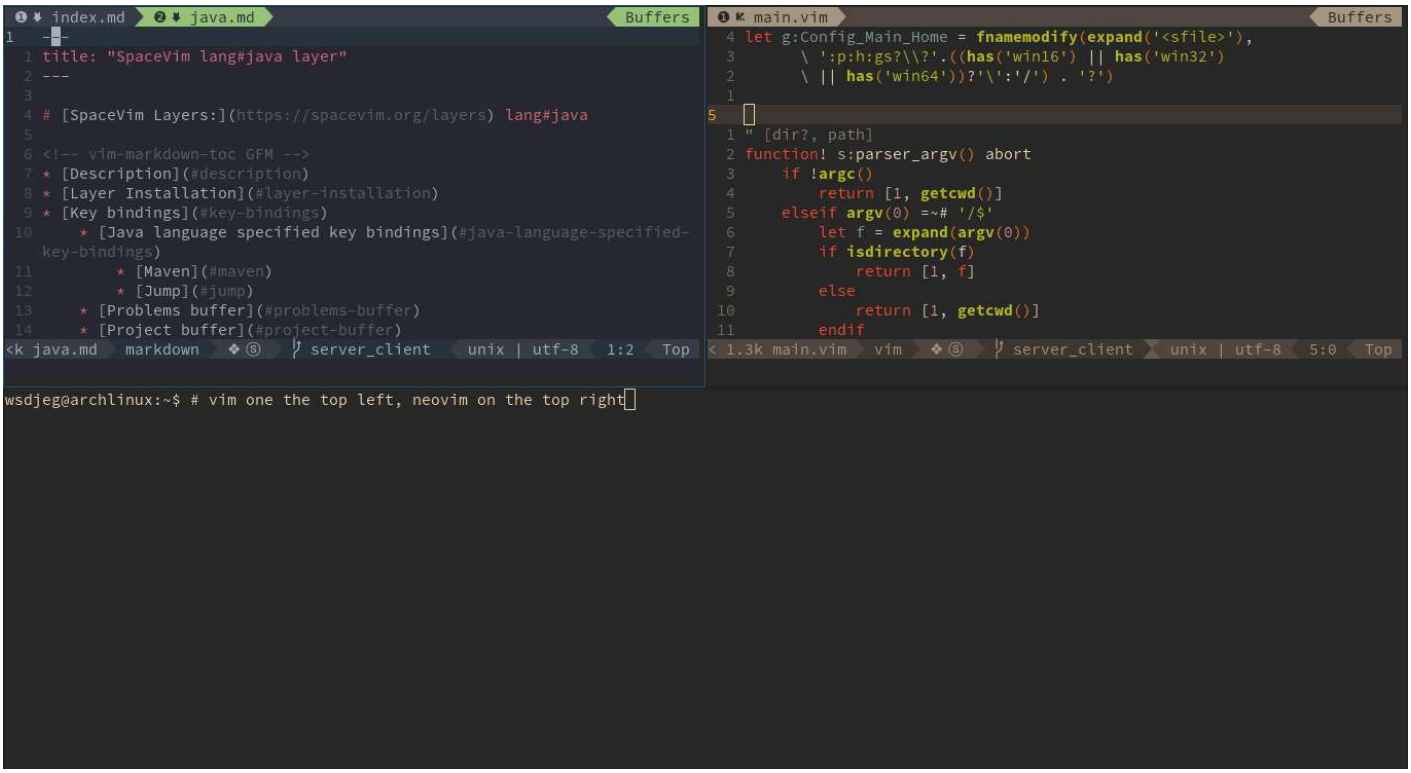
SpaceVim starts a server at launch. This server is killed whenever you close your Vim windows.

Connecting to the Vim server

If you are using Neovim, you need to install neovim-remote, then add this to your bashrc.

```
export PATH=$PATH:$HOME/.SpaceVim/bin
```

Use `svc` to open a file in the existing Vim server, or use `nsvc` to open a file in the existing Neovim server.



Powered by Jekyll