Use Language Servers for Development in the FreeBSD Src Tree

Table of Contents

1. Introduction . . . 1
2. Requirements . . . 1
3. Editor settings . . . 2
4. Compilation database . . . 4
5. Final . . . 6

1. Introduction

This guide is about setting up a FreeBSD src tree with language servers performing source code indexing. The guide describes the steps for Vim/NeoVim and VSCode. If you use a different text editor you can use this guide as a reference and search the equivalent commands for your preferred editor.

2. Requirements

In order to follow this guide we need to install certain requirements. We need a Language server, `ccls` or `clangd`, and optionally a compilation database.

The installation of the Language server can be performed via `pkg` or via ports. If we chose `clangd` we need to install `llvm`.

Using `pkg` to install `ccls`:

```
# pkg install ccls
```

If we want to use `clangd` we need to install `llvm` (The example command uses `llvm15` but choose the version you desire):

```
# pkg install llvm15
```

To install via ports choose a favorite combination of tools from each category below:

- Language server implementations

- devel/ccls
- devel/llvm12 (Other versions are okay, but newer is better. Replace `clangd12` with clangdN in case other versions are used.)

- Editors
  - editors/vim
  - editors/neovim
  - editors/vscode
- Compilation database generator
  - devel/python (For llvm's scan-build-py implementation)
  - devel/py-pip (For rizsotto's scan-build implementation)
  - devel/bear

3. Editor settings

3.1. Vim/Neovim

3.1.1. LSP client plugins

The built-in plugin manager is used for both editors in this example. The LSP client plugin used is prabirshrestha/vim-lsp.

To set up the LSP client plugin for Neovim:

```
# mkdir -p ~/.config/nvim/pack/lsp/start
# git clone https://github.com/prabirshrestha/vim-lsp
~/.config/nvim/pack/lsp/start/vim-lsp
```

and for Vim:

```
# mkdir -p ~/.vim/pack/lsp/start
# git clone https://github.com/prabirshrestha/vim-lsp ~/.vim/pack/lsp/start/vim-lsp
```

To enable the LSP client plugin in the editor, add the following snippet into ~/.config/nvim/init.vim when using Neovim, or ~/.vim/vimrc when using Vim:

For ccls

```
au User lsp_setup call lsp#register_server({
    \ 'name': 'ccls',
    \ 'cmd': {server_info->['ccls']},
    \ 'allowlist': ['c', 'cpp', 'objc'],
    \ 'initialization_options': {
```

```
    \     'cache': {
    \         'hierarchicalPath': v:true
    \     }
    \ }})
```

For clangd

```
au User lsp_setup call lsp#register_server({
    \ 'name': 'clangd',
    \ 'cmd': {server_info->['clangd15', '--background-index', '--header-
insertion=never']},
    \ 'allowlist': ['c', 'cpp', 'objc'],
    \ 'initialization_options': {},
    \ })
```

Depending on the version that you installed for `clangd` you might need to update the `server-info` to point to the correct binary.

Please refer to https://github.com/prabirshrestha/vim-lsp/blob/master/README.md#registering-servers to learn about setting up key bindings and code completion. The official site of clangd is https://clangd.llvm.org, and the repository link of ccls is https://github.com/MaskRay/ccls/.

Below are the reference settings of keybindings and code completions. Put the following snippet into ~/.config/nvim/init.vim, or ~/.vim/vimrc for Vim users to use it:

```
function! s:on_lsp_buffer_enabled() abort
    setlocal omnifunc=lsp#complete
    setlocal completeopt-=preview
    setlocal keywordprg=:LspHover

    nmap <buffer> <C-]> <plug>(lsp-definition)
    nmap <buffer> <C-W>] <plug>(lsp-peek-definition)
    nmap <buffer> <C-W><C-]> <plug>(lsp-peek-definition)
    nmap <buffer> gr <plug>(lsp-references)
    nmap <buffer> <C-n> <plug>(lsp-next-reference)
    nmap <buffer> <C-p> <plug>(lsp-previous-reference)
    nmap <buffer> gI <plug>(lsp-implementation)
    nmap <buffer> go <plug>(lsp-document-symbol)
    nmap <buffer> gS <plug>(lsp-workspace-symbol)
    nmap <buffer> ga <plug>(lsp-code-action)
    nmap <buffer> gR <plug>(lsp-rename)
    nmap <buffer> gm <plug>(lsp-signature-help)
endfunction

augroup lsp_install
    au!
    autocmd User lsp_buffer_enabled call s:on_lsp_buffer_enabled()
augroup END
```

3.2. VSCode

3.2.1. LSP client plugins

LSP client plugins are required to launch the language server daemon. Press `Ctrl+Shift+X` to show the extension online search panel. Enter `llvm-vs-code-extensions.vscode-clangd` when running clangd, or `ccls-project.ccls` when running ccls.

Then, press `Ctrl+Shift+P` to show the editor commands palette. Enter `Preferences: Open Settings (JSON)` into the palette and hit `Enter` to open settings.json. Depending on the language server implementations, put one of the following JSON key/value pairs in settings.json:

For clangd

```
[
    /* Begin of your existing configurations */
    ...
    /* End of your existing configurations */
    "clangd.arguments": [
        "--background-index",
        "--header-insertion=never"
    ],
    "clangd.path": "clangd12"
]
```

For ccls

```
[
    /* Begin of your existing configurations */
    ...
    /* End of your existing configurations */
    "ccls.cache.hierarchicalPath": true
]
```

4. Compilation database

A Compilation database contains an array of compile command objects. Each object specifies a way of compiling a source file. The compilation database file is usually compile_commands.json. The database is used by language server implementations for indexing purpose.

Please refer to https://clang.llvm.org/docs/JSONCompilationDatabase.html#format for details on the format of the compilation database file.

4.1. Generators

4.1.1. Using scan-build-py

4.1.1.1. Installation

`intercept-build` tool from scan-build-py is used to generate compilation database.

Install devel/python to get python interpreter first. To get `intercept-build` from LLVM:

```
# git clone https://github.com/llvm/llvm-project /path/to/llvm-project
```

where /path/to/llvm-project/ is your desired path for the repository. Make an alias in the shell configuration file for convenience:

```
alias intercept-build='/path/to/llvm-project/clang/tools/scan-build-py/bin/intercept-build'
```

rizsotto/scan-build can be used instead of LLVM's scan-build-py. The LLVM's scan-build-py was rizsotto/scan-build merged into the LLVM tree. This implementation can be installed by `pip install --user scan-build`. The `intercept-build` script is in ~/.local/bin by default.

4.1.1.2. Usage

In the top-level directory of the FreeBSD src tree, generate the compilation database with `intercept-build`:

```
# intercept-build --append make buildworld buildkernel -j`sysctl -n hw.ncpu`
```

The `--append` flag tells the `intercept-build` to read an existing compilation database (if a compilation database exists) and append the results to the database. Entries with duplicated command keys are merged. The generated compilation database by default is saved in the current working directory as compile_commands.json.

4.1.2. Using devel/bear

4.1.2.1. Usage

In the top-level directory of the FreeBSD src tree, to generate compilation database with `bear`:

```
# bear --append -- make buildworld buildkernel -j`sysctl -n hw.ncpu`
```

The `--append` flag tells `bear` to read an existing compilation database if it is present, and append the results to the database. Entries with duplicated command keys are merged. The generated compilation database by default is saved in the current working directory as compile_commands.json.

5. Final

Once the compilation database is generated, open any source files in the FreeBSD src tree and LSP server daemon will be launched as well in background. Opening source files in the src tree for the first time takes significantly longer time before the LSP server is able to give a complete result, due to initial background indexing by the LSP server compiling all the listed entries in the compilation database. The language server daemon however does not index the source files not appearing in the compilation database, thus no complete results are shown on source files not being compiled during the `make`.