Use o Vim para desenvolvimento C # no Linux

fonte

Se você é um usuário ávido do Vim, provavelmente tentará usar o editor de terminais para *tudo* relacionado ao texto.
Nesta postagem, mostrarei como obter suporte conveniente para C # com o Vim no Linux.

C # é tradicionalmente uma daquelas linguagens de programação que lucram com um IDE, um ambiente de desenvolvimento integrado.

O Vim ainda pode ser uma alternativa viável se você precisar de recursos mínimos, como definições de tipo ou preenchimento automático.

Instale o NET.Core

Como usuário do Arch Linux, meu primeiro instinto é instalar pacotes com o gerenciador de pacotes do Arch.
De alguma forma, isso parece conflito com o servidor de idiomas que instalaremos posteriormente.

Portanto, recomendo uma instalação manual. O Arch Linux wiki explica como. As instruções também funcionam para outras distribuições.

1. Baixe o dotnet-install.sh script para Linux.
2. Execute o script para a versão estável:

```
chmod +x dotnet-install.sh
./dotnet-install.sh --install-dir /usr/share/dotnet -channel LTS -
version latest
```

(Você pode precisar `sudo` porque o usuário normal não possui permissões para o `/usr/share/dotnet` pasta.)

Instale o Language Server

Precisamos **OmniSharp Roslyn**, uma implementação de servidor de idioma entre plataformas.

O README do projeto está densamente cheio de informações. A originalidade tentou construir o executável do zero, porque isso é destaque. Mas é *não* necessário e pode levar à frustração.

Vá para o guia de liberações e escolha uma versão adequada antes da construção.

Por exemplo, faça o download da versão 1.37.5 para Linux de 64 bits com enrolar e extraí-lo para `$HOME/.bin` pasta:

```
curl -sL https://github.com/OmniSharp/omnisharp-roslyn/releases/download/v1.37.5/omnisharp-linux-x64.tar.gz | tar xvzf - -C ~/home/.bin
```

Instale o LSP

Vim precisa de um plug-in para o Protocolo do servidor de idiomas.

Eu uso **prabirshrestha / vim-lsp**, uma implementação assíncrona que funciona tanto no Vim 8 quanto no NeoVim. O plug-in usa VimL e, portanto, não possui dependências externas.

Instale com suporte a pacotes nativos ou um gerenciador de plugins de sua escolha. Exemplo:

```
cd ~/vim/pack
git submodule init
git submodule add https://github.com/prabirshrestha/vim-lsp.git
git add .gitmodules vim/pack/prabirshrestha/vim-lsp
git commit
```

Agora registre o OmniSharp Language Server. Copiei minha configuração de um artigo de um colega blogueiro de tecnologia (hauleth.dev). Criei um novo arquivo na minha pasta Vim (`~/vim/plugin/lsp.vim`) com o seguinte conteúdo:

```
func! s:setup_ls(...) abort
    let l:servers = lsp#get_whitelisted_servers()

    for l:server in l:servers
        let l:cap = lsp#get_server_capabilities(l:server)

        if has_key(l:cap, 'completionProvider')
            setlocal omnifunc=lsp#complete
        endif
```

```
        if has_key(l:cap, 'hoverProvider')
            setlocal keywordprg=:LspHover
        endif

        if has_key(l:cap, 'definitionProvider')
            nmap <silent> <buffer> gd <plug>(lsp-definition)
        endif

        if has_key(l:cap, 'referencesProvider')
            nmap <silent> <buffer> gr <plug>(lsp-references)
        endif
    endfor
endfunc

augroup LSC
    autocmd!
    autocmd User lsp_setup call lsp#register_server({
                \ 'name': 'omnisharp-roslyn',
                \ 'cmd': {_->[&shell, &shellcmdflag, 'mono
$HOME/.bin/omnisharp/OmniSharp.exe --languageserver']},
                \ 'whitelist': ['cs']
                \})

    autocmd User lsp_server_init call <SID>setup_ls()
    autocmd BufEnter * call <SID>setup_ls()
augroup END
```

Nota: Você não precisa criar um novo arquivo para a configuração, é claro. Basta encontrar uma maneira de adicionar as configurações ao Vim / NeoVim (por exemplo, via `init.vim` configuração).

Nota: Se você instalou o OmniSharp em um diretório diferente do que `$HOME/.bin`, você precisa ajustar o `cmd` seção.

`&shell` e `&shellcmdflag` são específicos para vim-lsp (e não é realmente necessário no Linux):

> Recomenda-se usar o & shell com o & shellcmdflag ao executar arquivos de script que podem ser executados especialmente em janelas onde os arquivos _.bat e _.cmd não podem ser iniciados sem executar o shell primeiro. Isso é comum para executável instalado por npm para nodejs.

`mono` é o utilitário que permite executar `.exe` arquivos no Linux. Deve estar na sua máquina graças ao .Instalação do NET Core.

Agora, assim que você abrir um arquivo com filetype `cs` (para C #), o servidor de idiomas entrará em ação automaticamente.
Você pode digitar `K` (no modo normal) quando você passa o mouse sobre uma palavra-chave e obtém algumas informações sobre a palavra no cursor.

Destacamento da sintaxe

Destacamento da sintaxe funciona fora da caixa com o Tempo de execução do Vim.

Bônus: Formatação

Não consegui encontrar uma solução sancionada para formatar o C#. Por enquanto, estou usando **Não crustify**, um embelezador de código para idiomas no estilo C.

Esta ferramenta não é específica do Vim. Eu corro do terminal ou via comando shell externo em Vim.

Instale um binário pré-compilado de GitHub ou use o gerenciador de pacotes do seu sistema operacional.

Você pode personalizar o Uncrustify ao seu gosto e precisa de um arquivo de configuração padrão.

Aqui estão minhas configurações (`~/.uncrustify.cfg`):

```
#
# Formatter for c#, java, etc.
#

newlines = LF       # AUTO (default), CRLF, CR, or LF

indent_with_tabs        = 0     # 1=indent to level only, 2=indent
with tabs
input_tab_size          = 8     # original tab size
output_tab_size         = 3     # new tab size
indent_columns          = output_tab_size
# indent_label          = 0     # pos: absolute col, neg: relative
column
indent_align_string     = False     # align broken strings
indent_brace            = 0
indent_class            = true

nl_start_of_file            = remove
```

```
# nl_start_of_file_min      = 0
nl_end_of_file              = force
nl_end_of_file_min          = 1
nl_max                      = 4
nl_before_block_comment     = 2
nl_after_func_body          = 2
nl_after_func_proto_group   = 2

nl_assign_brace         = add       # "= {" vs "= \n {"
nl_enum_brace           = add       # "enum {" vs "enum \n {"
nl_union_brace          = add       # "union {" vs "union \n {"
nl_struct_brace         = add       # "struct {" vs "struct \n {"
nl_do_brace             = add       # "do {" vs "do \n {"
nl_if_brace             = add       # "if () {" vs "if () \n {"
nl_for_brace            = add       # "for () {" vs "for () \n {"
nl_else_brace           = add       # "else {" vs "else \n {"
nl_while_brace          = add       # "while () {" vs "while () \n
{"
nl_switch_brace         = add       # "switch () {" vs "switch ()
\n {"
nl_func_var_def_blk     = 1
nl_before_case          = 1
nl_fcall_brace          = add       # "foo() {" vs "foo()\n{"
nl_fdef_brace           = add       # "int foo() {" vs "int
foo()\n{"
nl_after_return         = TRUE
nl_brace_while          = remove
nl_brace_else           = add
nl_squeeze_ifdef        = TRUE

pos_bool            = trail     # BOOL ops on trailing end

eat_blanks_before_close_brace   = TRUE
eat_blanks_after_open_brace = TRUE


mod_paren_on_return         = add       # "return 1;" vs "return
(1);"
mod_full_brace_if           = add       # "if (a) a--;" vs "if (a)
{ a--; }"
mod_full_brace_for          = add       # "for () a--;" vs "for ()
{ a--; }"
mod_full_brace_do           = add       # "do a--; while ();" vs
"do { a--; } while ();"
```

```
mod_full_brace_while        = add       # "while (a) a--;" vs
"while (a) { a--; }"

sp_before_byref                 = remove
sp_before_semi                  = remove
sp_paren_paren                  = remove   # space between (( and
))
sp_return_paren                 = remove   # "return (1);" vs
"return(1);"
sp_sizeof_paren                 = remove   # "sizeof (int)" vs
"sizeof(int)"
sp_before_sparen                = force        # "if (" vs "if("
sp_after_sparen                 = force        # "if () {" vs "if
(){"
sp_after_cast                   = remove   # "(int) a" vs "(int)a"
sp_inside_braces                = force        # "{ 1 }" vs "{1}"
sp_inside_braces_struct         = force        # "{ 1 }" vs "{1}"
sp_inside_braces_enum           = force        # "{ 1 }" vs "{1}"
sp_inside_paren                 = remove
sp_inside_fparen                = remove
sp_inside_sparen                = remove
sp_inside_square                = remove
#sp_type_func                   = ignore
sp_assign                       = force
sp_arith                        = force
sp_bool                         = force
sp_compare                      = force
sp_assign                       = force
sp_after_comma                  = force
sp_func_def_paren               = remove   # "int foo (){" vs "int
foo(){"
sp_func_call_paren              = remove   # "foo (" vs "foo("
sp_func_proto_paren             = remove   # "int foo ();" vs "int
foo();"
sp_func_class_paren             = remove
sp_before_angle                 = remove
sp_after_angle                  = remove
sp_angle_paren                  = remove
sp_angle_paren_empty            = remove
sp_angle_word                   = ignore
sp_inside_angle                 = remove
sp_inside_angle_empty           = remove
sp_sparen_brace                 = add
sp_fparen_brace                 = add
sp_after_ptr_star               = remove
```

```
sp_before_ptr_star               = force
sp_between_ptr_star              = remove

align_with_tabs              = FALSE     # use tabs to align
align_on_tabstop             = FALSE     # align on tabstops
align_enum_equ_span          = 4
align_nl_cont                = TRUE
align_var_def_span           = 1
align_var_def_thresh         = 12
align_var_def_inline         = TRUE
align_var_def_colon          = TRUE
align_assign_span            = 1
align_assign_thresh          = 12
align_struct_init_span       = 3
align_var_struct_span        = 99
align_right_cmt_span         = 3
align_pp_define_span         = 3
align_pp_define_gap          = 4
align_number_right           = TRUE
align_typedef_span           = 5
align_typedef_gap            = 3
align_var_def_star_style     = 0

cmt_star_cont           = TRUE
```

Acima você vê as opções de um arquivo de configuração do GitHub com alterações feitas de esta questão.

Pensamentos

Usar o Vim para o desenvolvimento do C # é um pouco instável. Tive melhor sucesso com idiomas como Go ou OCaml. Mas em uma pitada, ele funciona — mesmo no Linux.

Links

- Instale vários .Versões NET Core manualmente
- vim-lsp
- OmniSharp Roslyn
- Vim: Patógeno tão longo, olá carregamento de pacotes nativos
- Editor de Elixir VIsual (e editor iMproved)
- Não crustify