help - Vim日本語ドキュメント

紫　　= シェル上で実行するコマンド
ピンク = vim上で入力するコマンド
青　　= .vimrc に記述する設定(~/.vim/plugin/***.vim に記載でもOK)

【Concept】

★ 新規コマンドの作成（ヤンクしたデータの取得 + exコマンド）

現在のカーソル位置の文字列（単語）をヤンク
yiw

exコマンドで、<Ctrl+r>の後に、<”> を入力
（現在のカーソル位置の文字列をコピーして、、、続けてexコマンドで何かを行う）
:<C-r>”

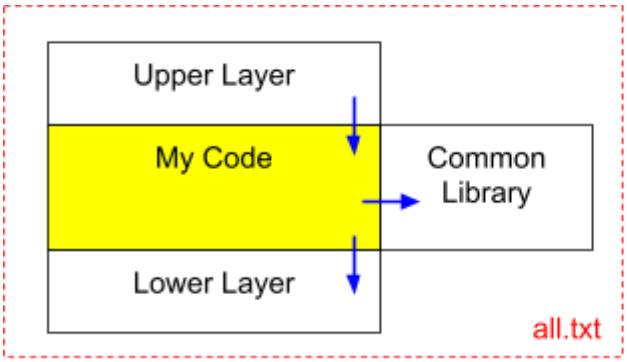
検索した文字列の取得
:<C-r>/

ヤンクしたデータとexコマンドを組み合わせて、入力を自動化したものをmapで割り当てる
使用頻度が高く、利用価値の高いコマンドを .vimrc に map しておく

★ 大量のコードをどうやって読み書きするか？（all.txt ）

あらかじめ自分の見るべきファイルをリストアップしておく
このリストされたファイル(all.txt)経由で、見たいファイル(*.c, *.h)を開く
ターミナルから vim xx/yy/zz.c などとはいちいちやらない！
このリスト化されたファイルの一覧をtag-jump, grep の対象とする（IDEのワークスペース）

膨大なディレクトリ群から見るべき対象を絞り込み、一覧を生成する（生成方法は何でもOK）
ディレクトリ名が長い場合は、フォルダを掘って、その中からシンボリックリンクを使う
自分の担当範囲と共通ライブラリ、下位層、上位層当たりを一覧にしておく
$ find dir1/ -name “*.[ch]” > all.txt
$ find dir2/ -name “*.cpp” >> all.txt



関係するコードは全てall.txtに記載し、いつでもボタン一発でgrep, tag-jumpできるようにしておく

all.txtを開いて、縦分割して、左側に一覧を表示。
自分が開きたいファイルをvim上で検索する。/file<CR> とすることで、目標物に素早くたどり着く。ファイルブラウザ、コンソール画面からファイルを探す必要は無い。
カーソルを左において、<Ctrl+f>でファイルを右側に開く（左側には一覧がキープされる）
左側に開く場合は、<Shift+f>、*.c を見ているときに、all.txtに戻るときは、<Shift-A>。

```
$ vim all.txt
:vs
/hoge.c
map <C-f> 0y$<C-w>l:open <C-r>"<CR>
map F 0y$:open <C-r>"<CR>
map A :open all.txt<CR>
```

「左側に一覧、右側にコード」、のように並べて表示すると見やすくなる
左側でメモをとりつつ、右側にコードを表示するということもできる
all.txt に新しいファイル名を追記して、<Ctrl+f>で新規ファイルとしてオープンできる

★ カーソル位置のファイルを開く

<gf>でカーソル位置のファイルを開くことができる。上記の<F>と同じ。
systemのインクルードファイル(stdio.h, unistd.hなど)を<gf>で開くことができる
（後述の、<Ctrl+b>で戻れる）
<Ctrl+w>gf で新しいタブでファイルを開く。<Ctrl+w>f だと水平分割して開く

★ all.txt を新しいタブで再度開く

新しいタブを開いて、all.txt を開いて、新しいファイルを開く、、、これは何度も繰り返される作業なので、ボタン一発で出来た方がよい
<Ctrl+a>により、新しいタブの左側にall.txt, 右側に現在のファイルが再表示される

```
map <C-a> mA:tabnew all.txt<CR>:vs<CR><C-W>25<<C-W>l'Azz
```

このコマンドは、「あるファイルを参照しているときに、現在のポジションをキープしたまま、同ファイル内の別の箇所を参照する」、という目的にも使える。
上記の<Ctrl+a>で新しいタブを開いた場合、:lopenすると、location-listは継承されるが、以前のタブとは別物として扱うことが出来る

★ フィルタ結果を左に表示する

新しいタブに、選択した文字列のgrep結果を左側に表示する、<ff>
あらかじめ<Ctrl+v>で選択した状態から、<ff>を入力する
選択した文字列を含まない一覧を作る場合は、<fv>

```
map ff y:MyFilter <C-R>" -h  <CR><C-W>25<
map fv y:MyFilter <C-R>" -hv <CR><C-W>25<

if !&cp && has("user_commands")
command! -nargs=* MyFilter call MYFILTER (<f-args>)
function! MYFILTER (v1, v2)
    execute ":w! ~/.vim/hoge.tmp"
    execute ":!grep " . a:v1 . " " . a:v2 . " ~/.vim/hoge.tmp > ~/.vim/hoge.grep"
```

```
    execute ":tabnew"
    execute ":r ~/.vim/hoge.grep"
    execute ":vs"
endfunction
endif
```

上記のスクリプトは、開いているファイルに含まれる文字列を対象としている。
all.txt だけに使えるわけではない。ソースコード内のあるプレフィックスを含む文字列を左に表示することもできる。

【Move, Search】

★ カーソルの移動、タブの移動、マウス

上<k>, 下<j>, 左<h>, 右<l> のキーバインディングを、タブ、ウィンドウの移動にも適用する
<Ctrl+l>で左のタブ、<Ctrl+h>で右のタブに移動する

```
:tabnew hoge.c
map <C-l> :tabnext<CR>
map <C-h> :tabprev<CR>
```

縦に分割する場合は、:vs, 水平に分割する場合は:sp
<Ctrl+j>で分割したウィンドウの下、<Ctrl+k>で上に移動する

```
:vs
:sp
map <C-j> <C-w>j
map <C-k> <C-w>k
```

<Ctrl+w>h, <Ctrl+w>l で縦分割したウィンドウを左右に移動する

マウスでクリックした位置にカーソル移動する（タブも選択できる）

```
set mouse=a
```

TeraTermで文字列を選択時にコピペをしたい場合は、:set mouse= にすることでコピーできるようになる

分割したウィンドウの幅調整
<F9>, <F10>で縦幅、<F11>, <F12>で横幅を調整する
（縦幅調整後はカーソルは下ウィンドウ、横幅調整後は右ウィンドウになる）

```
map <F9> <C-j><C-W>3+
map <F10> <C-j><C-W>3-
map <F11> <C-w>l<C-W>3>
map <F12> <C-w>l<C-W>3<
```

ファイルの先頭<gg>、ファイルの末尾<G>、行の先頭<0>、行の末尾<$>に移動

関数の先頭に後方移動<{{>、前方移動<}}>

対応する括弧、#if ~ #endif に移動<%>

★ 以前のカーソル位置に戻る、進む
デフォルトはo,i だがやや覚えにくいので、割り当てを変えている。
既存コマンドを再割り当てするときは、noremapを使う
戻るとき(back)は<Ctrl+b>、進む(next)ときは、<Ctrl+n>
```
noremap <C-b> <C-o>
noremap <C-n> <C-i>
```

ファイルを開いたときに以前のカーソル位置を復旧させる
```
:au BufReadPost * if line("'\"") > 1 && line("'\"") <= line("$") | exe "normal! g'\"" | endif
```

★ 検索
カーソル位置で<*>で、単語の境界有りで前方検索。<n>で続けて前方検索、<N>だと後方
単語境界有りの場合は、/\<***\> のように、括る

検索した文字の色づけのON/OFF
```
set hlsearch
set nohlsearch
```

インクリメンタル検索しない
```
set noincsearch
```

大文字小文字を区別して検索する
```
set noignorecase
```

現在のカーソル位置から、単語の末尾までを検索対象とする、<Shift+y>
```
map Y ye/<C-R>"<CR>
```

現在のハイライトを非表示にする、<q>
```
map q :noh<CR>
```

前回検索した文字に加えて新たに検索文字をORで追加する、<F3>
複数のキーワードをハイライト表示したいときに使用する
ハイライトされるキーワードは全てのタブ、ウィンドウで同じである
```
map <F3> yiw/<Up>\|\<<C-R>"\><CR>
```

左側のカーソル位置にある単語を、右側で検索する、<F7>
<F8>と使用方法はほぼ同じ。<F7>の後に、<n>を連打で、検索を継続できる
関数定義ではなく、関数の参照箇所を検索する場合はこちらのほうがよい
```
map <F7> yiw<C-w>l/\<<C-R>"\><CR>
```

★ tag-jump
all.txt が存在するディレクトリに、tagsファイルを生成しておく。
```
$ ctags `cat all.txt`
```

$ vim all.txt

tag-jumpする場合は、<Ctrl+]>、戻る場合は<Ctrl+T>
画面を水平に分割しつつtag-jumpをするときは、<Ctrl+w><Ctrl+]>
map <C-t> <C-t>zz
map <C-]> <C-]>zz

指定したキーワードに対して、tag-jumpをする
名前はTABで補完可能。stagだと画面分割有り（vtagはない）
:tag hoge_func
:stag hoge_func

tag-jump先が複数ある場合（同じ名前の定義が複数存在する）
<Ctrl+]>
:ts
番号を選択して、<Enter>

タグファイルを更新する、<Shift+T>
map T :!ctags `cat all.txt`<CR>

新しいタブを開いてtag-jumpする、<Ctrl+o>
map <C-o> yiw:tabnew <CR>:tag <C-R>"<CR>zz

現在のカーソル位置から単語末尾までに対して、tag-jumpする、<Ctrl+y>
map <C-y> ye:tag <C-R>"<CR>zz

縦分割状態で、左側のキーワードを右側でtag-jumpして開く、<F8>
左側でメモファイル、ヘッダファイルを開きつつ、右側でソースを開く、のように使う。
コードがちゃんと階層化されているなら、左側が関数の呼び出し側、右側が呼び出される側、のようにすると非常に見やすい。
map <F8> yiw<C-W>l:tag <C-R>"<CR>zz

△注意
tag-jump出来ない場合は、all.txt とは別ファイルで定義されている、もしくは糞マクロにより定義部がパースできていない、外部のライブラリに定義がある、、等が考えられる。
マクロが原因の場合は、ctagsでパースできないようなコードは書くべきでは無い（メリットよりもデメリットの方が多い）

★ grep(vim 内部grep)
grepの対象をall.txtに記載されているファイル全てとする
見つからない場合は、all.txtのリストを見直す
grepprgであらかじめ外部コマンドを割り当てておく
set grepprg=Grep2

Grep2コマンドはall.txtの全ての要素に対して、grepを仕掛けるだけでOK
$ cat ~/bin/Grep2

```
#!/bin/sh
cat all.txt |xargs grep -nE --color $@
```

カーソル位置にある単語を境界ありで、lgrepする、<F6>
grep結果を参照するときは、:lopen（省略すると、:lop）
```
map <F6> yiw:lgrep -w <C-R>"<CR>:lop<CR>
```

選択した範囲をlgrepする、<F5>
```
map <F5>   y:lgrep    <C-R>"<CR>:lop<CR>
```

★ lfile, cfile(vim 外部grep)

lfile, cfileは外部コマンドで行った、grep出力結果をvimに取り込む（ファイル名、行番号が含まれていること）。
取り込んだ結果に対して<Enter>もしくは、ダブルクリックでジャンプできる(:set mouse=a)。
“l”は各タブごとに確保されるが、”c”は全体で共有される。基本的には ”l” を使う。
lfileで先頭の要素にジャンプし、lopenで一覧(location-list)を画面下に表示する

```
$ Grep2 aaa | grep -v bbb > hoge.txt
:lfile hoge.txt
:lopen
```

★ 開いているファイル内のgrep

grepprgはGrep2が既に設定されているので、外部grepを使用するMygrepを作る
（毎回、grepprgを再設定するという選択肢もあり）

カーソル位置の単語を、現在開いているファイルのみでlgrepする、<Ctrl+g>
```
map <C-g> yiw:Mygrep <C-R>"<CR>

if !&cp && has("user_commands")
command! -nargs=* Mygrep call MYGREP(<f-args>)
function! MYGREP (pattern)
   let tmp = ":!grep -EHn " . a:pattern . " " . expand("%p") .  " > ~/.vim/ttt"
   execute tmp
   unlet tmp
   execute ":lfile ~/.vim/ttt"
   execute ":lopen"
endfunction
endif
```

★ 指定したファイルをall.txtから探す(include fileを探す)

カーソル位置にあるファイルをall.txt から探して、左側に表示する、<F2>
拡張子までを含めてヤンクするために<b>で単語の先頭に戻っている
（単語の先頭文字で<F2>を入力できない）
違うディレクトリに同じファイルがある場合は、location-listに複数表示される
```
map <F2> b<C-v>eeey<C-w>l:open all.txt<CR>:Mygrep <C-r>"<CR><C-k><C-f>
```

★ シンボルの一覧を表示
選択した文字列を含むシンボルをtagsファイルから検索して左側に表示、<F4>
シンボル名、ファイル名が左側に表示される。<F8>で移動する
map <F4> y:!TagParse.sh <C-R>"<CR>:tabnew<CR>:vs<CR><C-W>25<:r ~/.vim/hoge.tmp<CR>$<F8>zz

$ cat ~/bin/TagParse.sh
#!/bin/sh
cat tags | awk '{print $1}' | grep $1 | uniq -c > ~/.vim/hoge.tmp

候補が複数ある場合は、数字が2以上になる（uniq -c の結果）

【Edit】

◆ 入力補完
インサートモードでキーワードを途中まで入力してから、<Ctrl+n>, <Ctrl+p>
次の候補に移動するときは、<Ctrl+n>, 戻るときは、<Ctrl+p>。
補完を完了するときは<ESC>で抜ける

◆ Undo/Redo
<u>でUndo、<Ctrl-r>でRedo
Undoすると、1つ前の編集位置に戻る（カーソル位置を元の位置に戻す目的でも使える）

◆ 以前の編集操作を繰り返す
何か編集してから、<.>（ピリオド）
「移動コマンドの後に、<.>」を連続して交互に押して高速に処理できる
（<j>で下に移動して、<.>で貼り付け、、を繰り返す、等）

◆ 範囲選択
<Shift+v>のあと上下移動で、1行ずつ選択範囲を増やせる
<Ctrl+v>だと、矩形選択が可能。1行の途中まで選択も可能
<v>だと矩形選択はできないが、1文字ずつ選択範囲を広げられる（行の途中から次の行の途中まで、が可能）
選択範囲が決まったら、<y>でヤンクして、<p>で貼り付ける。（<d>で切り取りも可能）

矩形選択された文字列は矩形としてペーストされる
<p>を押した位置を起点として、X,Y方向に展開される（足りない部分は空白が付与される）

現在の括弧の中全部（スコープ）を選択するコマンド
vi{

★ 文字列置換（現在編集中のファイル）
現在開いているファイルで、カーソル位置の単語に、単語境界ありで置換処理を行う、<S>
境界無しの場合は<s>

最後にエンターを押すまで置換処理は走らない。置換後の文字列を手動で設定してから<Enter>を押す（置換前の文字列をベースにして設定する）。
1行に処理を繰り返して行う場合は、/g を付与する

```
map s yiw:1,$s/<C-R>"/<C-R>"
map S yiw:1,$s/\<<C-R>"\>/<C-R>"
:1,$s/\<before\>/after/g
:1,$s/\<before\>/after
:1,$s/before/after/g
:1,$s/before/after
```

行数を指定する場合は(100行目から200行目）

```
:100,200s/before/after
```

★ 文字列置換（複数ファイル）

location-listに表示されている項目に対して、置換処理を行う、<c>
特定のキーワードをあらかじめgrepして、マッチした部分全てを置換する

```
:MyConvert before after
map c yiw:MyConvert  \<<C-R>"\>  <C-R>"

if !&cp && has("user_commands")
command! -nargs=* MyConvert call MYCONVERT(<f-args>)
function! MYCONVERT (v1, v2)
    let loclist = getloclist(0)
    let target = "!@#$"
    let filenum = 0
    for list in loclist
        "echo bufname(list.bufnr) ':' list.lnum '=' list.text
        if bufname(list.bufnr) !=# target
            if filenum !=# 0
                execute ":w"
            endif
            execute ":open " bufname(list.bufnr)
            let filenum = filenum + 1
        endif

        let tmp = ':' . list.lnum . ',' . list.lnum . 's/' . a:v1 . '/' . a:v2 . '/'
        execute tmp
        unlet tmp
        let target = bufname(list.bufnr)
    endfor

    execute ":w"
    unlet loclist
    unlet target
    unlet filenum
endfunction
```

```
endif
```

lgrepの結果(location-list) を編集（置換前に不要なものを削除する）
modifiableでgrep結果が編集可能となり、lbufferで編集結果が反映される

```
:set modifiable
dd
:lbuffer
```

★ Copy/Paste（プロセス跨いだコピペ）

タブ、ウィンドウを跨いでコピペする場合は、<yy>, <p> でOK
ターミナル1のvimからターミナル2のvimへコピペを行う場合は、<yy>, <Ctrl+c>, <Ctrl+p>
手順は以下の3つ（yy, Ctrl+c までがターミナル1、Ctrl+p を別のターミナル2で実行）

```
何かしら文字列をヤンクする<yy>
<C-c> :!rm -f ~/.vim/ttt<CR>:new ~/.vim/ttt<CR>p:wq<CR>
<C-p> :r ~/.vim/ttt<CR>
```

当然ながら、上記のやり方では、ホストを跨いだコピペは出来ない
TeraTermで、コピペを行う場合は、インサートモードにして、set mouse=の状態（aをはずす）で、貼り付けを行うこと。
ファイルを ~/tmp/hoge.txt 配下にコピーして、それを :r ~/tmp/hoge.txt で読み取れば良いだけ。

★ Diff

diff -rq で差分の一覧を表示させ、そのファイルをvimで開く。
差分を見たいファイルにカーソルを合わせて、:Diff

```
$ diff -rq before/ after/ > hoge.diff
$ vim hoge.diff
カーソルを上下に移動(Files aaa.c and bbb.c differ のリストから選ぶ)
:Diff
```

差分が記述されている箇所はカーソルを合わせることが出来る（削除された領域はカーソルを合わせることは出来ない）
差分を適用する場合は<dp>、差分を受け取る（削除する）場合は<do>。
下の差分に移動する場合は<]c>、上の差分に移動する場合は<[c>
編集中に差分を更新して再描画する場合は、:diffu
Undo/Redoは普段通り使える。all.txt と同じディレクトリで、差分ファイルを開けば、普段と同じ作業がそのまま可能となる

```
if !&cp && has("user_commands")
  command -range=% Diff :call DIFF()
  func DIFF()
    let line = getline(".")
    let mx='Files\s\(\f\+\)\sand\s*\(\f\+\)'
    let l = matchstr(line, mx)
    let file1 = substitute(l, mx, '\1', '')
    let file2 = substitute(l, mx, '\2', '')
    let exe1 = "tabnew " . file2
```

```
    let exe2 = ":vert diffsplit " . file1

    execute ":set scrolloff=3"
    execute exe1
    execute ":set syntax=off"
    execute exe2
    execute ":set syntax=off"
    execute ":set diffopt=filler,iwhite"

    unlet line
    unlet mx
    unlet l
    unlet file1
    unlet file2
    unlet exe1
    unlet exe2
  endfunc
endif
```

set diffopt によって、Diffの設定を変えられる。
上記の場合は削除領域を黒で埋める(filler)、空白の差分を無視する(iwhite)。

【etc】

★ カーソル位置にラインを引く

```
set cursorline
hi CursorLine      cterm=none ctermbg=246
```

set cursorline しておかないと、hi CursorLine は有効にならない

★ 外部コマンド(git/svn/make) 連携

現在作業中のリポジトリに対して、Diffをとる、<Q>
all.txt を開いているディレクトリがリポジトリのトップである場合は単純だが、リポジトリが複数ある場合は、細工が必要

```
Q :!svn diff > hoge.diff<CR>:tabnew<CR>:r hoge.diff<CR>:set syntax=diff<CR>
```

Make環境のトップディレクトリにall.txt を配置しておいて、makeもvimから行う、<M>

```
M :lmake<CR>:lopen<CR>
```

★ man結果の取り込み

カーソル位置のキーワードに対して、man を行うのが、<Shift+k>である。
この結果はデフォルトではlessに渡されており、vimの操作ができないため実は不便。
以下の設定で、新規タブに結果を取り込める（新規タブから、引数の型、必要な includeファイル名等をコピペできる）

```
map K yiw:!man.pl <C-R>" <CR>:tabnew<CR>:r ~/.vim/mmm<CR>:set
syntax=man<CR>/<C-R>"<CR>

$ cat ~/bin/man.pl
#!/usr/bin/perl

my $arg = shift;
my $buf = `man 2 $arg`;
if(length($buf) == 0) {
    $buf = `man 3 $arg`;
}
open(FH, "> $ENV{HOME}/.vim/mmm") or die;
print FH $buf;

close(FH);
```

出力結果を取り込むときは、:tabnew してから :r で読み込むのが安パイ
vim上でtempファイルを開かないことでバッティングを防げる（連続で実行可能となる）

★ 拡張子の関連付け
*.txt のファイルを開いたら、filetype=txt として解釈する
~/.vim/syntax/txt.vim を作成すれば、syntaxを独自に定義できる
```
:au BufRead,BufNewFile *.conf set filetype=conf
:au BufRead,BufNewFile *.txt set filetype=txt
```

★ ステータス
ステータスラインを常に表示する(=2)。0は常に表示しない。1だとウィンドウが複数の場合のみ
```
set laststatus=2
```

★ **256色対応**
colorscheme で設定した名前と同じファイル(~/.vim/colors/256_hoge.vim)を用意しておく。
カラー設定は詳細に決められるので、好きなように記載する。
```
set t_Co=256
colorscheme 256_hoge
```

/usr/share/vim/XXX/syntax の配下に、大量の***.vim ファイルがある。
ここで各プログラム言語で使用されているSyntaxの設定がある。
これを.vimrc でオーバーライドできる。パラメータの名前はこのファイルから探せばOK。

主な設定一覧
```
$ cat ~/.vim/colors/256_hoge.vim
" NOTE: this is not the original file! converted for use with xterm-256

"------------ for ALL -------------
hi Folded term=standout ctermbg=Black ctermfg=DarkCyan
hi FoldColumn term=standout ctermbg=Black ctermfg=DarkCyan
```

```
hi Search  term=standout ctermbg=Yellow ctermfg=Black
set hlsearch
set background=light
"hi Normal          ctermfg=Black ctermbg=252
"hi clear

hi Normal          ctermfg=Black ctermbg=247
hi LineNr          ctermfg=238 ctermbg=245
hi Cursor          ctermfg=White ctermbg=Black
hi CursorLine      cterm=none ctermbg=246
"hi CursorLine      cterm=underline
hi CursorLineNr    ctermfg=White ctermbg=26

hi TabLineSel      ctermfg=White ctermbg=Blue

"------------ for *.diff -------------
hi diffOnly    ctermfg=Black
"hi diffFile   ctermbg=191 ctermfg=26 cterm=bold
hi diffAdded   ctermfg=28
hi diffRemoved ctermfg=165

"hi Search  cterm=bold ctermbg=190 ctermfg=232
hi DiffChange term=standout ctermbg=248 ctermfg=21
hi DiffText   term=standout ctermbg=117 ctermfg=21
hi DiffDelete term=standout ctermbg=232 ctermfg=159
hi DiffAdd    term=standout ctermbg=159 ctermfg=232

"------------ for Makefile -------------
hi makeIdent  ctermfg=20
hi makeTarget ctermfg=93 cterm=bold

"------------ for javaScript -------------
"syn keyword javaScriptIdentifier       arguments this var
hi javaScriptIdentifier ctermfg=20 cterm=bold
hi javaScriptFunction   ctermfg=20 cterm=bold
hi javaScriptBraces          ctermfg=20
hi javaScriptParens          ctermfg=20

"------------ for C, etc,, -------------
"hi Normal          ctermfg=Black ctermbg=146
hi Type            ctermfg=26 cterm=bold
hi Comment         ctermfg=28

hi Number          ctermfg=201
hi Operator        ctermfg=20 cterm=bold
```

```
hi Label           ctermfg=20 cterm=bold
hi Conditional     ctermfg=20 cterm=bold
hi Statement       ctermfg=20 cterm=bold
"hi Keyword         ctermfg=33 cterm=bold
hi String          ctermfg=160
hi PreProc         ctermfg=93

hi cHoge           ctermfg=32
hi Todo            ctermfg=Yellow ctermbg=Black cterm=bold

hi Visual          ctermfg=117 ctermbg=Black
hi StatusLine      ctermbg=152 ctermfg=26 cterm=bold
"hi StatusLineNC    ctermfg=236 ctermbg=252

hi cssClassName    ctermfg=Black
hi cssBraces       ctermfg=Black
hi cssIdentifier   ctermfg=Black
hi cssFunctoinName ctermfg=Black
hi htmlTag         ctermfg=Black
hi htmlEndTag      ctermfg=Black
hi xmlTag          ctermfg=Black
hi xmlTagName      ctermfg=Black
hi xmlEndTag       ctermfg=Black

hi Special         ctermfg=Black

hi helpHyperTextJump    ctermfg=20 cterm=bold
hi helpHyperTextEntry   ctermfg=20 cterm=bold
hi helpVim              ctermfg=20 cterm=bold

hi vimFuncName          ctermfg=26 cterm=bold
hi vimVar               ctermfg=Black
"hi Identifier          ctermfg=26 cterm=bold

"
"hi Visual          ctermfg=236 ctermbg=210 cterm=reverse
"hi Title           ctermfg=15 ctermbg=236 cterm=bold
"hi Boolean         ctermfg=181 cterm=bold
"hi Character       ctermfg=181 cterm=bold
"hi Comment         ctermfg=8
"hi Constant        ctermfg=181 cterm=bold
"hi Debug           ctermfg=181 cterm=bold
"hi Define          ctermfg=223 cterm=bold
"hi Delimiter       ctermfg=245
"hi DiffAdd         ctermbg=239
"hi DiffChange      ctermbg=236
```

```
"hi DiffDelete      ctermfg=236 ctermbg=238 cterm=none
"hi DiffText        ctermfg=15 ctermbg=234 cterm=bold
"hi Directory       ctermfg=15 cterm=bold
"hi Error           ctermfg=0 ctermbg=14
"hi ErrorMsg        ctermfg=0 ctermbg=6
"hi Exception       ctermfg=123 cterm=underline
"hi Float           ctermfg=139
"hi FoldColumn      ctermfg=181 ctermbg=238
"hi Folded          ctermfg=181 ctermbg=236
"hi Function        ctermfg=228
"hi Identifier      ctermfg=15
"hi Include         ctermfg=223 cterm=bold
"hi IncSearch       ctermfg=0 ctermbg=131
"hi Keyword         ctermfg=15 cterm=bold
"hi Macro           ctermfg=223 cterm=bold
"hi ModeMsg         ctermfg=181 cterm=bold
"hi MoreMsg         ctermfg=15 cterm=bold
"hi NonText         ctermfg=234
"hi PreCondit       ctermfg=180 cterm=bold
"hi Question        ctermfg=15 cterm=bold
"hi Repeat          ctermfg=123 cterm=underline
"hi Search          ctermfg=0 ctermbg=131
"hi SpecialChar     ctermfg=181 cterm=bold
"hi SpecialComment  ctermfg=181 cterm=bold
"hi SpecialKey      ctermfg=8
"hi StorageClass    ctermfg=15 cterm=bold
"hi Structure       ctermfg=15 cterm=bold,underline
"hi Tag             ctermfg=181 cterm=bold
"hi Title           ctermfg=15 ctermbg=236 cterm=bold
"hi Typedef         ctermfg=15 cterm=bold,underline
"hi VertSplit       ctermfg=236 ctermbg=252
"hi Visual          ctermfg=236 ctermbg=210 cterm=reverse
"hi VisualNOS       ctermfg=236 ctermbg=210 cterm=bold,underline
"hi WarningMsg      ctermfg=15 ctermbg=236 cterm=bold
"hi WildMenu        ctermfg=0 ctermbg=181
```

★ 折りたたみ

巨大な関数があり全体を把握しにくいときは、折りたたみを使う
インデント単位で折りたたむ場合は、foldmethod=indent とする
折りたたみを開くのがzo, 閉じる場合はzc 、全て開く場合はzR、全て閉じるときはzM

```
:set foldmethod=indent
zc
zo
zR
zM
```

★ c++ のラムダ式を構文エラーと見なさない

```
let c_no_curly_error=1
```

★ Help参照

```
:help
:help usr_xx
```