More Instantly Better Vim

Damian Conway

I. Signs and Portents

It's More Of A Guideline, Actually

- I like to code to 80 columns, but I also like to have the option of "slopping over"
- So I no longer set `textwidth`
- But I still want to know when I'm breaking the normal boundaries
- Vim has an option for that:

```
"====[ Make the 81st column stand out ]====================

highlight ColorColumn ctermbg=magenta
set colorcolumn=81
```

- Except I find the solid line annoying
- So I use this instead:

```
"====[ Make the 81st column stand out ]====================

highlight ColorColumn ctermbg=magenta
call matchadd('ColorColumn', '\%81v', 100)
```

- By the way, `colorcolumn` also makes a terrific April Fools Day "gift"
- You could bury the following somewhere deep inside their `.vimrc`:

```
highlight ColorColumn ctermbg=red ctermfg=blue
exec 'set colorcolumn=' . join(range(2,80,3), ',')
```

Die Blinkënmatchen

- The `n` command
- *"Take me to the next match"*
- But it's not always easy to see where you've been taken
- Especially if you also use `set hlsearch`
- So maybe you need some kind of `set hlnext` as well?
- Start by mapping these…

```
"=====[ Highlight matches when jumping to next ]=============

nnoremap <silent> n   n:call HLNext(0.4)<cr>
nnoremap <silent> N   N:call HLNext(0.4)<cr>
```

- Then chose one of the following…

```
"=====[ Blink the matching line ]=============

function! HLNext (blinktime)
    set invcursorline
    redraw
    exec 'sleep ' . float2nr(a:blinktime * 1000) . 'm'
    set invcursorline
    redraw
endfunction
```

```
"=====[ Blink a red ring around the match ]=============

function! HLNext (blinktime)
    highlight RedOnRed ctermfg=red ctermbg=red
    let [bufnum, lnum, col, off] = getpos('.')
    let matchlen = strlen(matchstr(strpart(getline('.'),col-1),@/))
    echo matchlen
    let ring_pat = (lnum > 1 ? '\%'.(lnum-1).'l\%>'
            \ . max([col-4,1]) .'v\%<'.(col+matchlen+3).'v.\|' : '')
            \ . '\%'.lnum.'l\%>'.max([col-4,1]) .'v\%<'.col.'v.'
            \ . '\|'
            \ . '\%'.lnum.'l\%>'.max([col+matchlen-1,1])
            \ . 'v\%<'.(col+matchlen+3).'v.'
            \ . '\|'
            \ . '\%'.(lnum+1).'l\%>'.max([col-4,1])
            \ . 'v\%<'.(col+matchlen+3).'v.'
    let ring = matchadd('RedOnRed', ring_pat, 101)
    redraw
    exec 'sleep ' . float2nr(a:blinktime * 1000) . 'm'
    call matchdelete(ring)
    redraw
endfunction
```

```
"=====[ Briefly hide everything except the match ]=============

function! HLNext (blinktime)
    highlight BlackOnBlack ctermfg=black ctermbg=black
    let [bufnum, lnum, col, off] = getpos('.')
    let matchlen = strlen(matchstr(strpart(getline('.'),col-1),@/))
    let hide_pat = '\%<'.lnum.'l.'
            \ . '\|'
            \ . '\%'.lnum.'l\%<'.col.'v.'
            \ . '\|'
            \ . '\%'.lnum.'l\%>'.(col+matchlen-1).'v.'
            \ . '\|'
            \ . '\%>'.lnum.'l.'
    let ring = matchadd('BlackOnBlack', hide_pat, 101)
    redraw
    exec 'sleep ' . float2nr(a:blinktime * 1000) . 'm'
    call matchdelete(ring)
    redraw
endfunction
```

```
"=====[ Highlight the match in red ]=============

function! HLNext (blinktime)
    let [bufnum, lnum, col, off] = getpos('.')
    let matchlen = strlen(matchstr(strpart(getline('.'),col-1),@/))
    let target_pat = '\c\%#'.@/
    let ring = matchadd('WhiteOnRed', target_pat, 101)
    redraw
    exec 'sleep ' . float2nr(a:blinktime * 1000) . 'm'
    call matchdelete(ring)
    redraw
endfunction
```

3. Shady Characters

- I'm absolute about whitespace in code…
- Absolutely no tab characters
- Absolutely no other weird whitespace (e.g. non-breaking space)
- Absolutely no trailing whitespace at the end of a line
- Vim has two options that helps make these shady characters obvious
- `'list'` and `'listchars'`
- For example:

```
   set listchars=tab:>~,nbsp:_,trail:.
   set list
```

- Personally, I use:

```
exec "set listchars=tab:\uBB\uBB,trail:\uB7,nbsp:~"
    set list
```

II. Applied Laziness

The Jung and the Shiftless

- One of the commonest, but most awkward, key chords when using Vim is `<SHIFT>;`
- Otherwise known as: `:`
- That `<SHIFT>` is just plain annoying, especially as the unshifted `;` isn't even doing anything particularly interesting
- So I steal it for initiating Ex commands:

```
nnoremap  ;  :
```

- Or, if you're one of the five people on the planet who actually do use `;` you could simply swap the two keys:

```
nnoremap  ;  :
nnoremap  :  ;
```

- I do the same kind of key-swap for Visual mode too
- I *never* want regular Visual mode
- I *always* want Visual Block mode
- So that one should be easier to type:

```
nnoremap    v   <C-V>
nnoremap <C-V>     v

vnoremap    v   <C-V>
vnoremap <C-V>     v
```

Patch your diff's syntax

- I'm not a fan of syntax colouring, as I generally find the results distracting instead of helpful
- Too many channels of too much information
- However, occasionally syntax colouring really *does* improve understanding
- In particular, when editing diffs and patchfiles
- You can request colouring in just these kinds of files with:

```
augroup PatchDiffHighlight
    autocmd!
    autocmd BufEnter  *.patch,*.rej,*.diff   syntax enable
augroup END
```

- Or, using the "filetype" mechanism:

```
filetype on
augroup PatchDiffHighlight
    autocmd!
    autocmd FileType  diff   syntax enable
augroup END
```

This file isn't big enough for the two of us!

- I'm a multi-terminal developer
- So I'm forever seeing this:

```
Swap file "~/tmp/.temporary_file.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (Q)uit, (A)bort:
```

- I noticed that, from the options listed, I only ever chose 'Q' (if the same file was already open in another terminal)
- ...or 'O' (if it was a remnant swapfile from some earlier disaster)

- So I made ‘O’ the default (with a warning message):

```
augroup NoSimultaneousEdits
    autocmd!
    autocmd SwapExists * let v:swapchoice = 'o'
    autocmd SwapExists * echomsg ErrorMsg
    autocmd SwapExists * echo 'Duplicate edit session (readonly)'
    autocmd SwapExists * echohl None
    autocmd SwapExists * sleep 2
augroup END
```

- And if I want ‘Q’ this time, I just reflexively hit ‘ZZ’
- Later, I realized that half of the time my workflow with this new feature was to immediately quit, then search through a dozen other terminal windows for the Vim session with that file already open
- So I wrote a plugin called `autoswap_mac` that does that for me too
- Sadly, it currently only works on a Mac
- And only if you:

```
set title titlestring=
```

- But you’re very welcome to hack it for other platforms!

III. Cunning Linguistics

"I've got a little list"

- I write a lot of documentation in Vim
- One thing I really hate is getting half way through typing a text list, only to realize it should have been a bullet list
- So I automated the transformation in a plugin called `listtrans.vim`
- I map it in both Normal and Visual modes:

```
nmap  ;l  :call ListTrans_ToggleFormat()
vmap  ;l  :call ListTrans_ToggleFormat('visual')
```

"¡No hablo (mucho) Inglés!"

- Many of my clients are not native English speakers
- So, for them, I always try to avoid difficult English and stick to "Basic English"
- To assist with that, I created a custom spelling configuration: `en-basic.latin1.spl`
- And, naturally, defined a suitable mapping to enable it:

```
nmap  ;s     :set invspell spelllang=en<CR>
nmap  ;ss    :set    spell spelllang=en-basic<CR>
```

- You can build your own spelling lists just as easily
- For example, to build my "Basic English" spelling list:

```
:mkspell  ~/.vim/spell/en-basic  basic_english_words.txt
```

Accent-uate the positive

- Don't you just love that Vim has no trouble with non-ASCII characters?
- The only problem is: I can never remember the digraph sequences required to type them
- What I need is to have `<CTRL-K>` automatically show me that list each time:

```
inoremap <C-K> <C-O>:digraphs<CR><C-K>
```

- *Except that doesn't actually work!*
- You need to factor it out into a function:

```
inoremap <expr> <C-K> ShowDigraphs()

function! ShowDigraphs ()
    digraphs
    call getchar()
    return "\<C-K>"
endfunction
```

- Now it works, but it's distracting
- And interrupts my flow
- I decided that what I really needed was a proper "heads-up" display
- So I created the `hudigraphs.vim` plugin
- But that only highlighted the *real* problem: the digraph shortcuts themselves suck!
- So I created a new heads-up digrapher plugin: `betterdigraphs.vim`
- With new digraph sequences that make sense (to me, at least)

IV. Visual Acuity

The fear of all sums

- I'm surprised how often I need to add up a list of numbers
- Typically, too many to just enter them into my on-screen calculator
- But not enough that I could be bothered firing up Gnumeric or Numbers or Excel
- Or even: `https://github.com/vim-scripts/spreadsheet.vim`
- All I want is to type in a column of numbers, then have Vim add it up
- So I wrote a plugin *(surprise, surprise!)*
- `vmath.vim` sums, averages, and finds extremities
- I make it available in both Normal and Visual modes:

```
vmap <expr>  ++  VMATH_YankAndAnalyse()
nmap         ++  vip++
```

- Once you've computed the sum, it's available in the default yank buffer (so you can immediately paste it with the usual `p`). The sum is also in the `"s` buffer (e.g. `"sp`).
- The average is available in `"a`, the minimum value in `"n`, the maximum value in `"x`, and the min-to-max range as a single string is available in `"r`

The surprising convenience of a blocked colon

- I love Visual Block mode
- Except when my visual blocks don't act like visual blocks
- Normally colon commands issued from within a Visual mode affect the entire lines covered by the Visual selection
- Which is handy…except where it's a damn nuisance
- That's where the `vis.vim` plugin can help:

  `http://vim.sourceforge.net/scripts/script.php?script_id=1195`

- It provides the magic `:B` command
- Short for: "apply to ***B***lock only"

It's a drag

- This next one was actually a "viewer submission"
- After last year's talk, Steve Matney showed me a way cool trick
- Dragging visual blocks around:

```
vnoremap  K  xkP`[V`]
vnoremap  U  xp`[V`]
vnoremap  L  >gv
vnoremap  H  <gv
```

- I liked that so much that I wanted to eliminate the various edge-cases
- And add in a few other awesome cosmic powers
- So I created the `dragvisuals.vim` plugin
- You can map the various actions to anything you like
- I prefer:

```
vmap  <expr>  <LEFT>   DVB_Drag('left')
vmap  <expr>  <RIGHT>  DVB_Drag('right')
vmap  <expr>  <DOWN>   DVB_Drag('down')
vmap  <expr>  <UP>     DVB_Drag('up')
```

- My favourite feature is the "duplicator" option:

```
vmap  <expr>  D        DVB_Duplicate()
```

Conclusion

- No matter how well you already know Vim...
- ...there's always one more option to learn
- ...there's always one more behaviour to re-script
- ...there's always one more tool whose technological distinctiveness must be added to your own
- Because there's *always* one more annoyance to remove
- Grab the goodies from this talk:

  `http://tinyurl.com/IBV2013`

- Grab my entire `.vimrc` and plugin library from:

  `https://github.com/thoughtstream/Damian-Conway-s-Vim-Setup`

- Use them as-is (if you're keen enough)
- Retool them to your own needs (if you're brave enough)
- Recreate them in your own image (if you're crazy enough)
- Just remember to read the disclaimers!