Vim Mapathon

An advanced introduction to maps

Wolfgang Mehner

Vimfest 2017

23. September 2017

Let's start before the beginning

To get us started, *Abbreviations*:

```
abbreviate wtf water-tight ferrets
abbreviate !=  ~=
```

In Insert mode, type 1st argument, obtain the 2nd

Let's take a shortcut

Maps are more general:

```
inoremap (      ()<Left>

nnoremap <C-q> :wqall<CR>
nnoremap Gd    :split<CR>gd
```

Executes the right-hand side.
Can change modes, use special keys, . . .

Lessons to be learned

Is this interesting for the regular user?

$\rightarrow$ Yes, maps can hugely improve your workflow.
YOUR workflow.

Lessons to be learned

Is this interesting for plug-in developers?

$\rightarrow$ Of central importance.

Lessons to be learned

Is this interesting for plug-in developers?

→ Of central importance.

Maps: fast, integrate into the workflow - accessibility?
Menus: good overview - access not fast enough
Ex-commands: powerful and flexible - still not as fast

Lessons to be learned

Is this interesting for plug-in developers?

$\rightarrow$ Of central importance.

Maps: fast, integrate into the workflow - accessibility?
Menus: good overview - access not fast enough
Ex-commands: powerful and flexible - still not as fast

Combining all three: best usability and configurability

Make Vim great again?

Already is!

How can we beat the shell *for all our regular tasks*?

Make Vim great again?

Already is!

How can we beat the shell *for all our regular tasks*?

Provide often used commands via VimScript.
And match the comfort of the shell's tab-completion!

Make Vim great again?

Already is!

How can we beat the shell *for all our regular tasks*?

Provide often used commands via VimScript.
And match the comfort of the shell's tab-completion!

We may not beat an IDE for one specific language.

How can we provide consistent performance *for a dozen languages*?

Make Vim great again?

Already is!

How can we beat the shell *for all our regular tasks*?

Provide often used commands via VimScript.
And match the comfort of the shell's tab-completion!

We may not beat an IDE for one specific language.

How can we provide consistent performance *for a dozen languages*?

Provide accessible and configurable features!
Otherwise all the power we provide is lost.

Contents

Contents

VimScript

Vim's configuration language

Notes on VimScript:

- scripting language:
  dynamic data structures, dynamic typing, pass-by-reference . . .
- but with some unusual scoping:
  variables and functions linked to buffers, windows, scripts, . . .

VimScript
Filetype plug-ins

Filetype plug-ins:

- script in VimScript, especially for one filetype
- executed once for each new buffer

VimScript

Filetype plug-ins

Filetype plug-ins:

- script in VimScript, especially for one filetype
- executed once for each new buffer
- *e.g.* define buffer-local maps
- thus become filetype-specific maps

Buffer-Local Maps

Global maps:

```
inoremap ( ()<Left>
```

Local to C++ buffers:

```
inoremap <buffer> {<CR> {<CR>}<Esc>O
```

Mind the `noremap`.

Maps and Modes

Different modes warrant and require different maps:

```
inoremap (   ()<Left>
vnoremap ( s()<Esc>P

inoremap <buffer> {<CR>  {<CR>}<Esc>O
vnoremap <buffer> {<CR> S{<CR>}<Esc>Pk=iB
```

Calling for Help

Open a dictionary for the word under the cursor:

```
nnoremap Hen :call CallHelpWiki()<CR>
```

Calling for Help

Open a dictionary for the word under the cursor:

```
nnoremap Hen :call CallHelpWiki()<CR>
```

This is easiest with a bit of VimScript:

```
function CallHelpWiki ()
  let url  = "https://en.wiktionary.org/wiki/%s"
  let word = expand ( "<cword>" )

  let url_f = printf ( url, word )

  call system ( "firefox ".url )
endfunction
```

Implementation

```
function CallHelpWiki ()
  let url  = "https://en.wiktionary.org/wiki/%s"
  let word = expand ( "<cword>" )
  let word = substitute ( word, '\W', '', 'g' )
  if word == ""
    echomsg "no word under cursor"
    return
  endif
  let url_f = printf ( url, word )
  call system ( "firefox ".url_f )
endfunction
```

Expression Maps

Run Vim's grep.
(In Visual mode, put the selected word on the cmd.-line.)

```
nmap        <C-G>f        :grep  %<Left><Left>
imap        <C-G>f   <Esc>:grep  %<Left><Left>
vmap <expr> <C-G>f "<Esc>:grep ".@*." %<Left><Left>"
```

Expression Maps

Run Vim's grep.
(In Visual mode, put the selected word on the cmd.-line.)

```
nmap        <C-G>f       :grep   %<Left><Left>
imap        <C-G>f  <Esc>:grep   %<Left><Left>
vmap <expr> <C-G>f "<Esc>:grep ".@*." %<Left><Left>"
```

Maps with `<expr>`:
Evaluate the expression,
execute the result as the right-hand side:

```
"<Esc>:grep " . @* . " %<Left><Left>"
```

Expression Maps

Run Vim's `grep`.
(In Visual mode, put the selected word on the cmd.-line.)

```
nmap        <C-G>f      :grep   %<Left><Left>
imap        <C-G>f  <Esc>:grep   %<Left><Left>
vmap <expr> <C-G>f "<Esc>:grep ".@*." %<Left><Left>"
```

Maps with `<expr>`:
Evaluate the expression,
execute the result as the right-hand side:

```
 <Esc>:grep selectword %<Left><Left>
```

Contents

Command-Line Maps

We can define maps for the command-line, using `cmap`:

```
cnoremap <C-x>c \(\)<Left><Left>
cnoremap <C-x>w \<\><Left><Left>
```

In the same spirit as the brackets before.

Let's be more ambitious

My shell supports <Alt-Backspace> for deleting a whole word.
On the Vim cmd.-line, use <Ctrl-W>.

Let's code it ourselves as an exercise.

Let's be more ambitious

My shell supports <Alt-Backspace> for deleting a whole word.
On the Vim cmd.-line, use <Ctrl-W>.

Let's code it ourselves as an exercise.

Central trick:

```
cmap LHS <C-\>eEXPR<CR>
```

Evaluate `EXPR` and replace the command-line with the result.

Implementation

Use a function CmdLineWordDelete():

```
cmap LHS <C-\>eCmdLineWordDelete()<CR>
```

Interface:

```
function CmdLineWordDelete ()
  " ...
  return replacement_string
endfunction
```

Implementation

cont.

```
" current cmd.-line and position of the cursor
let cmdline = getcmdline ()
let cmdpos  = getcmdpos () - 1

" split:  <head><CURSOR><tail>
let cl_head = strpart ( cmdline, 0, cmdpos )
let cl_tail = strpart ( cmdline, cmdpos )

" replace 'cl_head'
" ...

" set new cmdline cursor position
call setcmdpos ( len(cl_head)+1 )

return cl_head.cl_tail
```

Implementation

cont.

```
" ...

" replace
if match ( cl_head, '\w$' ) > -1
  let cl_head = substitute ( cl_head, '\w\+$', '', '' )
else
  let cl_head = substitute ( cl_head, '.$', '', '' )
endif

" ...

return cl_head.cl_tail
```

What else can we do?

Provide <Ctrl-P> on the cmd.-line.

Same method as before.
But we need to cycle through different keyword matches.

Implementation

```
" current cmd.-line and position of the cursor
" split:  <head><CURSOR><tail>

if cl_head == b:CmdLineLast
  let b:CmdLineIndex  += 1     " next match
else
  let b:CmdLineMatches = ...   " search replacements
  let b:CmdLineIndex   = 0     " first match
endif

" modify cl_head
let cl_head = ... . b:CmdLineMatches[ b:CmdLineIndex ]

" set new cmdline cursor position

let b:CmdLineLast = cl_head   " saved for next call
return cl_head.cl_tail
```

The proper approach

Until now, public function:

```
function CmdLineWordDelete ()
  " ...
endfunction

cmap <c-bs> <C-\>eCmdLineWordDelete()<CR>
cmap <c-p>  <C-\>eCmdLineCompletion(-1)<CR>
cmap <c-n>  <C-\>eCmdLineCompletion(1)<CR>
```

The proper approach

Local implementation:

```
function s:CmdLineWordDelete ()
  " ...
endfunction

cmap <c-bs> <C-\>e<SID>CmdLineWordDelete()<CR>
cmap <c-p>  <C-\>e<SID>CmdLineCompletion(-1)<CR>
cmap <c-n>  <C-\>e<SID>CmdLineCompletion(1)<CR>
```

The proper approach

Local implementation and separate configuration:

```
function s:CmdLineWordDelete ()
  " ...
endfunction

cmap <Plug>MapsClWd <C-\>e<SID>CmdLineWordDelete()<CR>
cmap <Plug>MapsClCp <C-\>e<SID>CmdLineCompletion(-1)<CR>
cmap <Plug>MapsClCn <C-\>e<SID>CmdLineCompletion(1)<CR>
```

The proper approach

Local implementation and separate configuration:

```
cmap <c-bs> <Plug>MapsClWd
cmap <c-p>  <Plug>MapsClCp
cmap <c-n>  <Plug>MapsClCn

cmap <Plug>MapsClWd <C-\>e<SID>CmdLineWordDelete()<CR>
cmap <Plug>MapsClCp <C-\>e<SID>CmdLineCompletion(-1)<CR>
cmap <Plug>MapsClCn <C-\>e<SID>CmdLineCompletion(1)<CR>
```

Contents

Ex-Commands

Another powerful mechanism:
User-defined *ex-commands*.

Ex-Commands

Another powerful mechanism:
User-defined *ex-commands*.

Consider a make plug-in:

```
command -complete=file MakeFile
    :call <SID>SetMakeFile(<q-args>)
command -complete=customlist,<SID>MakeComplete Make
    :call <SID>Run(<q-args>)
```

Ex-Commands

Another powerful mechanism:
User-defined *ex-commands*.

Consider a `make` plug-in:

```
command -complete=file MakeFile
    :call <SID>SetMakeFile(<q-args>)
command -complete=customlist,<SID>MakeComplete Make
    :call <SID>Run(<q-args>)
```

We can provide custom tab-completion.

Tab-Completion

```
...  -complete=customlist,<SID>MakeComplete ...

function s:MakeComplete( ArgLead, CmdLine, CursorPos )
  let targets = s:GetMakeTargets( s:Makefile )
  return filter( copy( targets ), ...a:ArgLead...  ))
endfunction
```

Tab-Completion

```
...  -complete=customlist,<SID>MakeComplete ...

function s:MakeComplete( ArgLead, CmdLine, CursorPos )
  let targets = s:GetMakeTargets( s:Makefile )
  return filter( copy( targets ), ...a:ArgLead...  ))
endfunction
```

But this breaks tab-completions for filenames!

Tab-Completion

```
...  -complete=customlist,<SID>MakeComplete ...

function s:MakeComplete( ArgLead, CmdLine, CursorPos )
  let files   = split( glob( a:ArgLead."*" ), "\n" )
  let targets = s:GetMakeTargets( s:Makefile )

  return filter( copy( targets ), ...a:ArgLead... ) )
     + files
endfunction
```

Tab-Completion

```
...  -complete=customlist,<SID>MakeComplete ...

function s:MakeComplete( ArgLead, CmdLine, CursorPos )
  let files   = split( glob( a:ArgLead."*" ), "\n" )
  let targets = s:GetMakeTargets( s:Makefile )
  return filter( copy( targets ), ...a:ArgLead... ) )
     + files
endfunction
```

Sufficient for this use-case,
but Vim's filename completion is still nicer.

Contents

Calling for Help

cont.

Let's revisit this example:

```
function CallHelpWiki ()
  let url  = "https://en.wiktionary.org/wiki/%s"
  let word = expand ( "<cword>" )
  let word = substitute ( word, '\W', '', 'g' )
  if word == ""
    echomsg "no word under cursor"
    return
  endif
  let url_f = printf ( url, word )
  call system ( "firefox ".url_f )
endfunction
```

Calling for Help
cont.

Use a local function and an argument:

```
function s:CallHelp ( url )
  let url  = a:url
  let word = expand ( "<cword>" )
  let word = substitute ( word, '\W', '', 'g' )
  if word == ""
    echomsg "no word under cursor"
    return
  endif
  let url_f = printf ( url, word )
  call system ( "firefox ".url_f )
endfunction
```

Calling for Help

cont.

Provide an ex-command:

```
function s:CallHelp ( url )
  " ...
endfunction

command MapathonHelp :call <SID>CallHelp(<q-args>)

nnoremap Hen :MapathonHelp
  https://en.wiktionary.org/wiki/%s<CR>
nnoremap Hcp :MapathonHelp
  http://en.cppreference.com/mwiki/...search=%s<CR>
nnoremap Hqt :MapathonHelp
  http://qt-project.org/doc/qt-4.8/%s.html<CR>
```

Demonstration

Integrate some often used tools into Git.

Demonstration: Git, . . .