

# VIM for (PHP) Programmers

Andrei Zmievski

Open Source Fellow, Digg

Twitter: @a

<http://gravitonic.com/>

PHP Québec Conf ~ March 5, 2009

# help.

---

- ~ learn how to get help effectively
- ~ `:help` is your friend
- ~ use `CTRL-V` before a `CTRL` sequence command
- ~ use `i_` and `v_` prefixes to get help for `CTRL` sequences in `Insert` and `Visual` modes
- ~ use `CTRL-]` (jump to tag) and `CTRL-T` (go back) in help window

- ~ how well do you know vim's language?
- ~ what is the alphabet?
- ~ look at your keyboard
- ~ can you name what every key does?
- ~ modes - what are they?
- ~ how many do you know?
- ~ how many do you use?

if you don't like the language, change it

example: how do you quit vim quickly?

**ZZ** (exit with saving)

**ZQ** (exit without save)

or

```
:nmap ,w :x<CR>
```

```
:nmap ,q :q!<CR>
```

tip: set **showcmd** to see partial commands as you type them

# where am i?

---

How do you tell where you are?

- ~ simple - `CTRL-G`
- ~ detailed - `gCTRL-G`
- ~ do yourself a favor and `set ruler`
- ~ shows line, column, and percentage in status line
- ~ or configure it however you want with `'rulerformat'`

# moving.

---

- ~ do you use **h/j/k/l** for moving?
- ~ or are you stuck in GUI arrowy world?
- ~ if you are, re-learn
- ~ save yourself countless miles of movement between home row and arrows

How do you move to:

- ~ start/end of buffer? `gg` and `G`
- ~ line `n`? `nG` or `ngg`
- ~ `n%` into the file? `n%`
- ~ the first non-blank character in the line? `^`
- ~ first non-blank character on next line? `<CR>`
- ~ first non-blank character on previous line? `-`

# marks.

---

- ~ we can bookmark locations in the buffer
- ~ `m<letter>` sets mark named `<letter>` at current location
- ~ ``<letter>` jumps precisely to that mark
- ~ `'<letter>` jumps to the line with the mark
- ~ lowercase letter: mark is local to the buffer
- ~ uppercase letter: mark is global, your buffer will be switched to the file with the mark
- ~ `:marks` shows you your current marks



# marks.

---

- ~ marks are very handy for changing text
- ~ set a mark (let's say **ma**)
- ~ then you can do:
  - ~ **c`a** - change text from cursor to mark **a**
  - ~ **d`a** - delete text from cursor to mark **a**
  - ~ **=`a** - reformat lines from current one to the one with mark **a**

# marks.

- ~ let's say you jump somewhere
- ~ how do you go back?
- ~ `` `` moves you between the last two locations
- ~ you can set ``` (the context mark) explicitly:
  - ~ `m``, jump elsewhere, then come back with `` ``

tip: `CTRL-O` and `CTRL-I` move between positions in the full jump history, but can't be used as motions

`' .` and `` .` - jump to the line or exact location of the last modification

# insert.

---

- ~ **gi** - incredibly handy
- ~ goes to **Insert** mode where you left it last time
- ~ scenario: edit something, exit **Insert**, go look at something else, then **gi** back to restart editing

## Some more goodies:

- ~ **CTRL-Y** and **CTRL-E** (avoid work if you can)
  - ~ inserts chars from above or below the cursor
- ~ **CTRL-A** (oops, i want to do that again)
  - ~ inserts previously inserted text
- ~ **CTRL-R=<expr>** (built-in calculator)
  - ~ inserts anything vim can calculate
- ~ **CTRL-T** and **CTRL-D** (tab and de-tab)
  - ~ inserts or deletes one shiftwidth of indent at the start of the line

# delete.

---

set your `<Backspace>` free

```
:set backspace=start,indent,eol
```

lets you backspace past the start of edit, auto-indenting, and even start of the line

# search.

---

- ~ searching is essential
- ~ movement and information
- ~ how do you search?
- ~ f / F / t / T anyone?
- ~ how about \* and #?

# search

---

Search within the line:

- ~ **f/F<char>** jumps to the first **<char>** to the right/left and places cursor on it
- ~ **t/T<char>** jumps does the same, but stops one character short of it
- ~ **df;** - delete text from cursor to the first **;** to the right
- ~ **cT\$** - change text from cursor up to the first **\$** to the left

# search

---

- ~ often you want to find other instances of word under the cursor
  - ~ `*/#` - find next/previous instance of whole word
  - ~ `g*/g#` - find next/previous instance of partial word
- ~ or find lines with a certain word:
  - ~ `[I` and `]I` - list lines with word under the cursor
  - ~ more convenient to use a mapping to jump to a line:

```
:map ,f [I:let nr = input("Which one: ")<Bar>exe  
"normal " . nr . "[\t"<CR>
```



# search

---

- ~ of course, there's always regexp search
- ~ `/<pattern>` - search forward for `<pattern>`
- ~ `?<pattern>` - search backward for `<pattern>`
- ~ `n` repeats the last search
- ~ `N` repeats it in the opposite direction
- ~ vim regexp language is too sophisticated to be covered here

## Control your search options

- ~ `:set wrapscan` - to make search wrap around
- ~ `:set incsearch` - incremental search, `<Enter>` accepts, `<Esc>` cancels
- ~ `:set ignorecase` - case-insensitive search, or use this within the pattern:
  - ~ `\c` - force case-insensitive search
  - ~ `\C` - force case-sensitive search

# search.

---

- ~ remember that every search/jump can be used as a motion argument
- ~ `d/^#` - delete everything up to the next comment
- ~ `y/^class/;?function` - copy everything from current point to the first `function` before the first `class`

# replace.

---

- ~ `:[range]s/<pattern>/<replace>/{flags}`  
is the substitute command
- ~ used mainly with range addresses
- ~ range addresses are very powerful (read the manual)
- ~ but who wants to count out lines and do something like `:-23,'ts/foo/bar/`
- ~ in reality you almost always use a couple of shortcuts and **Visual** mode for the rest

# replace.

- ~ useful range addresses:
  - ~ % - equal to 1,\$ (the entire file)
  - ~ . - current line
  - ~ /<pattern>/ or ?<pattern>? - line where <pattern> matches
- ~ :%s/foo/bar/ - replace first foo in each matching line with bar in the entire file
- ~ :.,/<\body>/s,<br>,<br/>,gc - fix br tags from current line until the one with </body> in it, asking for confirmation (c - 'cautious' mode)

# replace.

---

- ~ **&** - repeat last substitution on current line
- ~ **:&&** - repeat it with the flags that were used
- ~ **g&** - repeat substitution globally, with flags

# text objects.

---

- ~ better know what they are
- ~ since they are fantastically handy
- ~ can be used after an operator or in **Visual** mode
- ~ come in “**inner**” and “**ambient**” flavors
- ~ inner ones always select less text than ambient ones

# text objects.

---

- ~ **aw, aW** - ambient word or WORD (see docs)
- ~ **iw, iW** - inner word or WORD (see docs)
- ~ **as, is** - ambient or inner sentence
- ~ **ap, ip** - ambient or inner paragraph
- ~ **a{, i{** - whole {...} block or text inside it
- ~ **a(, i(** - whole (..) block or just text inside it
- ~ **a<, i<** - whole <.> block or just text inside it



# text objects.

---

- ~ there are some cooler ones
- ~ `a'`, `i'` - single-quoted string or just the text inside
- ~ `a"`, `i"` - double-quoted string or just the text inside
  - ~ note that these are smart about escaped quotes inside strings
- ~ `at`, `it` - whole tag block or just text inside (HTML and XML tags)

# text objects.

---

## examples:

**das** - delete the sentence, including whitespace after

**ci(** - change text inside (..) block

**yat** - copy the entire closest tag block the cursor is inside

**gUi'** - uppercase text inside the single-quoted string

**vip** - select the paragraph in Visual mode, without whitespace after

# copy/delete/paste.

- ~ you should already know these
- ~ **y** - yank (copy), **d** - delete, **p** - paste after, **P** - paste before
- ~ **]p**, **]P** - paste after/before but adjust the indent
- ~ Useful mappings to paste *and* reformat/reindent

```
:nnoremap <Esc>P  P' [v' ]=  
:nnoremap <Esc>p  p' [v' ]=
```

# registers.

---

- ~ registers: your multi-purpose clipboard
- ~ you use them without even knowing
- ~ every **y** or **d** command copies to a register
- ~ unnamed or named
- ~ "**<char>** before a copy/delete/paste specifies register named **<char>**

# registers.

---

- ~ copying to uppercase registers append to their contents
  - ~ useful for picking out bits from the buffers and pasting as a chunk
- ~ "wyy - copy current line into register w
- ~ "WD - cut the rest of the line and append it to the contents of register W
- ~ "wp - paste the contents of register w
- ~ CTRL-Rw - insert the contents of register w (in Insert mode)

# registers.

---

- ~ you can record macros into registers
  - ~ `q<char>` - start recording typed text into register `<char>`
  - ~ next `q` stops recording
  - ~ `@<char>` executes macro `<char>`
  - ~ `@@` repeats last executed macro
- ~ use `:reg` to see what's in your registers

# undo.

---

- ~ original vi had only one level of undo
- ~ yikes!
- ~ vim has unlimited (limited only by memory)
- ~ set `'undolevels'` to what you need (1000 default)

# undo.

---

- ~ simple case: **u** - undo, **CTRL-R** - redo
- ~ vim 7 introduces branched undo
- ~ if you undo something, and make a change, a new branch is created
- ~ **g-**, **g+** - go to older/newer text state (through branches)



# undo.

---

- ~ you can travel through time
  - ~ **:earlier Ns,m,h** - go to text state as it was N seconds, minutes, hours ago
  - ~ **:later Ns,m,h** - go to a later text state similarly
- ~ **:earlier 10m** - go back 10 minutes, before I drank a can of Red Bull and made all these crazy changes. Whew.

# visual mode.

---

- ~ use it, it's much easier than remembering obscure range or motion commands
- ~ start selection with:
  - ~ **v** - characterwise,
  - ~ **V** - linewise
  - ~ **CTRL-V** - blockwise
- ~ use any motion command to change selection
- ~ can execute any normal or **:** command on the selection

# visual mode.

---

~ Visual block mode is awesome

~ especially for table-like text

tip: **o** switches cursor to the other corner,  
continue selection from there

~ Once you are in block mode:

~ **I<text><Esc>** - insert **<text>** before block on every line

~ **A<text><Esc>** - append **<text>** after block on every line

~ **c<text><Esc>** - change every line in block to **<text>**

~ **r<char><Esc>** - replace every character with **<char>**

# abbreviations

---

- ~ Real-time string replacement
- ~ Expanded when a non-keyword character is typed
  - ~ `:ab tempalte template` - fix misspellings
  - ~ more complicated expansion:
    - ~ `:iab techo <?php echo ?><Left><Left><Left>`

# windows.

---

- ~ learn how to manipulate windows
- ~ learn how to move between them
- ~ **:new**, **:sp** should be at your fingertips
- ~ **CTRL-W** commands - learn essential ones for resizing and moving between windows

# tab pages.

---

- ~ vim 7 supports tab pages
- ~ `:tabe <file>` to edit file in a new tab
- ~ `:tabc` to close
- ~ `:tabn`, `:tabp` (or `gt`, `gT` to switch)
- ~ probably want to map these for easier navigation (if `gt`, `gT` are too difficult)

# completion.

---

- ~ vim is very completion friendly
- ~ just use `<Tab>` on command line
  - ~ for filenames, set `'wildmenu'` and `'wildmode'` (I like `"list:longest,full"`)
  - ~ `:new ~/dev/fo<Tab>` - complete filename
  - ~ `:help 'comp<Tab>` - complete option name
  - ~ `:re<Tab>` - complete command
- ~ hit `<Tab>` again to cycle, `CTRL-N` for next match, `CTRL-P` for previous

# completion.

---

- ~ `CTRL-X` starts completion mode in Insert mode
- ~ follow with `CTRL-` combos (`:help ins-completion`)
- ~ i mostly use filename, identifier, and omni completion
- ~ when there are multiple matches, a nice completion windows pops up



# completion.

- ~ CTRL-X CTRL-F to complete filenames
- ~ CTRL-X CTRL-N to complete identifiers
- ~ hey, that's so useful I'll remap <Tab>

```
" Insert <Tab> or complete identifier
" if the cursor is after a keyword character
function MyTabOrComplete()
    let col = col('.')-1
    if !col || getline('.')[col-1] !~ '\k'
        return "\<tab>"
    else
        return "\<C-N>"
    endif
endfunction

inoremap <Tab> <C-R>=MyTabOrComplete(<><CR>
```

# completion.

---

- ~ omni completion is heuristics-based
- ~ guesses what you want to complete
- ~ specific to the file type you're editing
- ~ more on it later

# maps.

~ maps for every mode and then some

~ tired of changing text inside quotes?

```
:nmap X ci"
```

~ make vim more browser-like?

```
:nmap <Space> <PageDown>
```

~ insert your email quickly?

```
:imap ;EM me@mydomain.com
```

~ make `<Backspace>` act as `<Delete>` in Visual mode?

```
:vmap <BS> x
```

# options.

---

- ~ vim has hundreds of options
- ~ learn to control the ones you need
- ~ `:options` lets you change options interactively
- ~ `:options | resize` is better (since there are so many)

# sessions

---

- ~ a session keeps the views for all windows, plus the global settings
- ~ you can save a session and when you restore it later, the window layout looks the same.
- ~ `:mksession <file>` to write out session to a file
- ~ `:source <file>` to load session from a file
- ~ `vim -S <file>` to start editing a session

# miscellaneous

---

- ~ `gf` - go to file under cursor (`CTRL-W CTRL-F` for new window)
- ~ `:read` in contents of file or process
  - ~ `:read foo.txt` - read in `foo.txt`
  - ~ `:read !wc %:h` - run `wc` on current file and insert result into the text
- ~ filter text: `:%!sort`, `:%!grep`, or use `:!` in visual mode
  - ~ i like sorting lists like this: `vip:!sort`

# miscellaneous.

- ~ use command-line history
- ~ **:** and **/** followed by up/down arrows move through history
- ~ **:** and **/** followed by prefix and arrows restrict history to that prefix
- ~ **q:** and **q/** for editable history (**<Enter>** executes, **CTRL-C** copies to command line)

# miscellaneous

---

- ~ **CTRL-A** and **CTRL-X** to increment/decrement numbers under the cursor (hex and octal too)
- ~ **ga** - what is this character under my cursor?
- ~ **:set number** to turn line numbers on
- ~ or use this to toggle line numbers:  

```
:nmap <silent> <F6> set number!<CR>
```
- ~ **:set autowrite** - stop vim asking if you want to write the file before leaving buffer
- ~ **CTRL-E/CTRL-Y** - scroll window down/up without moving cursor



# miscellaneous

---

- ~ `:set scrolloff=N` to start scrolling when cursor is `N` lines from the top/bottom edge
- ~ `:set updatecount=50` to write swap file to disk after 50 keystrokes
- ~ `:set showmatch matchtime=3` - when bracket is inserted, briefly jump to the matching one
- ~ in shell: `fc` invokes vim on last command, and runs it after vim exits (or `fc N` to edit command `N` in history)
- ~ `vimdiff` in shell (`:help vimdiff`)

# miscellaneous.

---

- ~ map **CTRL-L** to piece-wise copying of the line above the current one

```
imap <C-L> @@@<ESC>hhkywj1?@@@<CR>P/@@@<CR>3s
```

# customization

- ~ customize vim by placing files in you `~/.vim` dir
- ~ filetype plugin on, filetype indent on

```
.vimrc - global settings
.vim/
  after/          - files that are loaded at the very end
    ftplugin/
    plugin/
    syntax/
    ...
  autoload/      - automatically loaded scripts
  colors/        - custom color schemes
  doc/           - plugin documentation
  ftdetect/      - filetype detection scripts
  ftplugin/      - filetype plugins
  indent/        - indent scripts
  plugin/        - plugins
  syntax/        - syntax scripts
```

# php: linting.

---

- ~ vim supports arbitrary build/lint commands
- ~ if we set 'makeprg' and 'errorformat' appropriately..
  - `:set makeprg=php\ -l\ %`
  - `:set errorformat=%m\ in\ %f\ on\ line\ %l`
- ~ now we just type `:make` (and `<Enter>` a couple of times)
- ~ cursor jumps to line with syntax error

# php: match pairs.

---

- ~ you should be familiar with `%` command (moves cursor to matching item)
- ~ used with `()`, `{}`, `[]`, etc
- ~ but can also be used to jump between PHP and HTML tags
- ~ use `matchit.vim` plugin
- ~ but `syntax/php.vim` has bugs and typos in the matching rule
- ~ i provide my own

# php: block objects.

- ~ similar to vim's built-in objects
  - ~ **aP** - PHP block including tags
  - ~ **iP** - text inside PHP block

## examples:

- ~ **vaP** - select current PHP block (with tags)
- ~ **ciP** - change text inside current PHP block
- ~ **yaP** - copy entire PHP block (with tags)
- ~ provided in my [.vim/ftplugin/php.vim](#) file

# php: syntax options.

- ~ vim comes with a very capable syntax plugin for PHP
- ~ provides a number of options
  - ~ `let php_sql_query=1` to highlight SQL syntax in strings
  - ~ `let php_htmlInStrings=1` to highlight HTML in string
  - ~ `let php_noShortTags = 1` to disable short tags
  - ~ `let php_folding = 1` to enable folding for classes and functions

# php: folding.

---

- ~ learn to control folding
  - ~ **zo** - open fold (if the cursor is on the fold line)
  - ~ **zc** - close closest fold
  - ~ **zR** - open all folds
  - ~ **zM** - close all folds
  - ~ **zj** - move to the start of the next fold
  - ~ **zk** - move to the end of the previous fold



# php: tags.

---

- ~ for vim purposes, tags are PHP identifiers (classes, functions, constants)
- ~ you can quickly jump to the definition of each tag, if you have a **tags** file
- ~ install Exuberant Ctags
- ~ it can scan your scripts and output **tags** file, containing identifier info
- ~ currently does not support class membership info (outputs methods as functions)
- ~ have to apply a third-party patch to fix

# php: tags.

- ~ use mapping to re-build tags file after editing

```
nmap <silent> <F4>  
  \ :!ctags -f ./tags  
  \ --langmap="php:+.inc"  
  \ -h ".php.inc" -R --totals=yes  
  \ --tag-relative=yes --PHP-kinds="+cf-v" .<CR>  
  
set tags=./tags,tags
```

- ~ all PHP files in current directory and under it recursively will be scanned

# php: tags.

---

- ~ **CTRL-]** - jump to tag under cursor
- ~ **CTRL-W CTRL-]** - jump to tag in a new window
- ~ **:tag <ident>** - jump to an arbitrary tag
- ~ **:tag /<regexp>** - jump to or list tags matching **<regexp>**
- ~ if multiple matches - select one from a list
- ~ **:tselect <ident> or /<regexp>** - list tags instead of jumping
- ~ **CTRL-T** - return to where you were
- ~ See also [taglist.vim](#) plugin

# php: completion.

---

- ~ vim 7 introduces powerful heuristics-based **omni** completion
- ~ **CTRL-X CTRL-O** starts the completion (i map it to **CTRL-F**)
- ~ completes classes, variables, methods in a smart manner, based on context

# php: completion.

---

- ~ completes built-in functions too
- ~ function completion shows prototype preview
  - ~ `array_<CTRL-X><CTRL-O>` shows list of array functions
  - ~ select one from the list, and the prototype shows in a preview window
  - ~ `CTRL-W CTRL-Z` to close preview window

# php: completion.

---

- ~ switches to HTML/CSS/Javascript completion outside PHP blocks
- ~ see more:
  - ~ `:help ins-completion`
  - ~ `:help popupmenu-completion`
  - ~ `:help popupmenu-keys`
  - ~ `:help ft-php-omni`

# plugins.

---

- ~ vim can be infinitely customized and expanded via plugins
- ~ there are thousands already written
- ~ installation is very easy, usually just drop them into `.vim/plugin`
- ~ read instructions first though

- ~ makes it possible to read, write, and browse remote directories and files
- ~ i usually use it over ssh connections via scp
- ~ need to run ssh-agent to avoid continuous prompts for passphrase
- ~ don't use passphrase-less keys!
- ~ once set up:
  - ~ `vim scp://hostname/path/to/file`
  - ~ `:new scp://hostname/path/to/dir/`



# NERDTree

---

- ~ similar to netrw browser but looks more like a hierarchical explorer
- ~ does not support remote file operations
  - ~ `:nmap <silent> <F7> :NERDTreeToggle<CR>`

# taglist.

---

- ~ provides an overview of the source code
- ~ provides quick access to classes, functions, constants
- ~ automatically updates window when switching buffers
- ~ can display prototype and scope of a tag
- ~ requires Exuberant Ctags

# taglist.

~ stick this in `~/.vim/after/plugin/general.vim`

```
let Tlist_Ctags_Cmd = "/usr/local/bin/ctags-ex"
let Tlist_Inc_Winwidth = 1
let Tlist_Exit_OnlyWindow = 1
let Tlist_File_Fold_Auto_Close = 1
let Tlist_Process_File_Always = 1
let Tlist_Enable_Fold_Column = 0
let tlist_php_settings = 'php;c:class;d:constant;f:function'
if exists('loaded_taglist')
    nmap <silent> <F8> :TlistToggle<CR>
endif
```

# snippetsEmu.

- ~ emulates some of the functionality of TextMate snippets
- ~ supports many languages, including PHP/HTML/CSS/Javascript
- ~ by default binds to **<Tab>** but that's annoying
- ~ need to remap the key after it's loaded
- ~ put this in `~/.vim/after/plugin/general.vim`

```
if exists('loaded_snippet')
    imap <C-B> <Plug>Jumper
endif
inoremap <Tab> <C-R>=MyTabOrComplete()<CR>
```

# php documentor.

- ~ inserts PHP Documentor blocks automatically
- ~ works in single or multi-line mode
- ~ doesn't provide mappings by default
- ~ read documentation to set up default variables for copyright, package, etc
- ~ put this in `~/.vim/ftplugin/php.vim`

```
inoremap <buffer> <C-P> <Esc>:call PhpDocSingle()<CR>i
nnoremap <buffer> <C-P> :call PhpDocSingle()<CR>
vnoremap <buffer> <C-P> :call PhpDocRange()<CR>
let g:pdv_cfg_Uses = 1
```

# project.

---

- ~ Provides IDE-like project file management
- ~ Lets you group files and access them quickly
- ~ Can grep through and execute custom commands

- ~ Tag-based search for a variety of information
- ~ Quick access to:
  - ~ buffers, files, windows, tabs
  - ~ objects, methods
  - ~ things from ctags database
  - ~ registers to paste text from
  - ~ current file changes to move to
  - ~ vim marks to jump to

# xdebug-ger.

---

- ~ allows debugging with xdebug through DBGp protocol
- ~ fairly basic, but does the job
- ~ vim needs to be compiled with **+python** feature
- ~ see resources section for documentation links



# vcsc`command`.

---

- ~ provides interface to CVS/SVN/git
- ~ install it, then `:help vcsccommand`

# conclusion.

---

- ~ vim rules
- ~ this has been only a partial glimpse
- ~ from my very subjective point of view
- ~ don't be stuck in an editor rut
- ~ keep reading and trying things out

# resources

---

- ~ vim tips: <http://www.vim.org/tips/>
- ~ vim scripts: <http://www.vim.org/scripts/index.php>
- ~ Exuberant Ctags: <http://ctags.sourceforge.net>
- ~ PHP patch for ctags: <http://www.live-emotion.com/memo/index.php?plugin=attach&refer=%CA%AA%C3%D6&openfile=ctags-5.6j-php.zip>
- ~ article on xdebug and vim: <http://2bits.com/articles/using-vim-and-xdebug-dbgp-for-debugging-drupal-or-any-php-application.html>
- ~ more cool plugins:
  - ~ Surround: [http://www.vim.org/scripts/script.php?script\\_id=1697](http://www.vim.org/scripts/script.php?script_id=1697)
  - ~ ShowMarks: [http://www.vim.org/scripts/script.php?script\\_id=152](http://www.vim.org/scripts/script.php?script_id=152)
  - ~ Vim Outliner: [http://www.vim.org/scripts/script.php?script\\_id=517](http://www.vim.org/scripts/script.php?script_id=517)
  - ~ Tetris: [http://www.vim.org/scripts/script.php?script\\_id=172](http://www.vim.org/scripts/script.php?script_id=172)

"As with everything, best not to look too deeply into this."

# Thank You!

<http://joind.in/121>

<http://digg.com/>

<http://gravitonic.com/talks/>