

Kruczki i sztuczki dla programujących w języku Perl korzystających z edytora Vim

OSZCZĘDZANIE PALCÓW

Edytor Vim potrafi obsłużyć szereg sztuczek, pozwalających oszczędzić nieco wpisywania z klawiatury. W tym artykule przyjrzymy się wybranym technikom, ułatwiającym życie osobom programującym w języku Perl.

MICHAEL SCHILLI

Zadna decyzja w życiu programisty nie jest tak ważna i nieodwracalna jak wybór edytora tekstów. Gdy ktoś raz zdecyduje się na Vi lub Emacs, z reguły pozostaje przy tym wyborze i będzie próbował wycisnąć maksimum wydajności ze swojego ulubionego narzędzia. Bardziej efektywne wykorzystanie edytora pozwala nie tylko zredukować ryzyko wystąpienia schorzeń wywołanych napięciem mięśni nadgarstka, lecz pozwala pisać szybciej i unikać błędów.

Podświetlanie składni

Podświetlanie słów kluczowych i konstrukcji języka w kodzie programu jest doskonałym mechanizmem wspomagającym pracę programisty, pozwalającym nieco oszczę-



Rysunek 1a: Fragment kodu w Perlu z wyłączonym...



Rysunek 1b: ...i włączonym podświetlaniem składni.

dzić oczy zmęczone godzinami wpatrywania się w monitor.

Jeśli edytowany jest zupełnie nowy plik bez odpowiedniego rozszerzenia nazwy oraz sekwencji wywołującej interpretera Perla, można edytorowi Vim wskazać, że ma do czynienia ze składnią Perla, wywołując polecenie `:set filetype=perl`.

Skróty

Programiści wierni jednemu językowi programowania zauważą po pewnym czasie, że pewne sekwencje znaków wpisują częściej od innych. Jako zagorzały zwolennik modułu `Log::Log4perl`, nie pamiętam, ile razy wpisywałem ciąg znaków `use`

`Log::Log4perl`

`qw (:easy);`. Na

szczęście Vim

pomógł mi

uproszczyć nieco

zadanie. Polecenie

`:abbreviate ul4p use`

`Log::Log4perl`

`qw (:easy);` <RETURN> definiuje `ul4p` jako skrót.

Gdy w trybie wpisywania

zostanie wpisany taki skrót, po którym nastąpi znak spacji lub przejścia do nowego wiersza, Vim automatycznie rozwinie dany ciąg znaków do odpowiedniego wywołania modułu `Log4perl`. Innym sposobem wprowadzania dłuższych sekwencji tekstowych za pomocą skrótów jest wstawianie zawartości pliku: `:abb ul4p <BACKSPACE> <ESC>:r ~/templ_4p <RETURN>`. To polecenie nakazuje, aby skrót `ul4p` powodował wstawienie w jego miejsce zawartości określonego pliku.



Rysunek 2a: Edytor makr w trybie rejestracji.



Rysunek 2b: Użytkownik po prostu dwukrotnie powtórza wykonania makra.

Makra klawiaturowe

Makra można wykorzystać między innymi do powtarzania częstych operacji modyfikujących różne, nieciągłe obszary tekstu. Rysunki 2a i 2b przedstawiają trzy nagłówki funkcji, które chciałbym wyodrębnić, wstawiając znaki komentarza (#).

W tym celu wymagane są następujące działania: najpierw wyszukiwany jest ciąg znaków `sub` za pomocą sekwencji `/sub`, następnie uruchamiany jest rejestrator makr `a`, wpisywane są znaki komentarza przed i po wierszu nagłówka funkcji i wyłą-

czany jest rejestrator makr. Następnie za pomocą *n* wyszukiwane jest kolejne wystąpienie ciągu znaków *sub* i powtarzane jest wywołanie makra za pomocą *@a*. Listę poleceń realizujących tę funkcję przedstawia Listing 1.

Wiersze ze znakami komentarza można wstawić również od razu przy zdefiniowaniu nowej funkcji. W tym celu można zdefiniować skrót klawiaturowy, przypisując go na przykład klawiszowi *F* w następujący sposób: `:map F o<ESC> 43i#<ESC>yvosub {<ENTER><ESC> Pk&}`. Teraz po naciśnięciu klawisza *F* w trybie poleceń, Vim wstawi nagłówek funkcji, po czym przełączy się w tryb wprowadzania i ustawi kursor w odpowiednim miejscu, oczekując na wprowadzenie nazwy nowej funkcji.

Dziwny zestaw literek w definicji *map* to jednoklawiszowe polecenia trybu poleceń edytora vi, starzy wyjadacze z pewnością je rozpoznają. Liczba znaków *#* wstawianych do wyodrębnienia nagłówka funkcji to kwestia gustu. W przykładzie stosuję 43. Mapowanie poleceń to doskonała technika w przypadku często powtarzających się sekwencji czynności, jak na przykład wpisywanie nagłówek funkcji. W ten sam sposób można

wpisywać inne często spotykane sekwencje, na przykład kod zbierający parametry funkcji `my (...) = @_;`.

Inną powszechnie stosowaną sekwencją poleceń jest zapisywanie pliku za pomocą `:w` oraz sprawdzanie poprawności składni skryptu `perl -c script.pl`. Przypiszmy wykonywanie tych zadań poleceniu *X* trybu poleceń edytora: `:nnoemap X :w<Enter>:!perl -c %<Enter>`.

Wcięcia

Dyskusje na temat zalet i wad zasady dokonywania wcięć w kodzie programów praktycznie nie mają końca. Gdzie umieszczać nawiasy klamrowe? O ile znaków należy wcinąć zagnieżdżony kod? Używać spacji czy znaków tabulacji? Każdy programista ma swoje preferencje, Vim umożliwia dostosowanie opcji do własnych potrzeb.

Wcięcia z użyciem znaków tabulacji to kwestia gustu. Wielu programistów unika ich z zasady. Gdy ustawimy opcję `:set expandtab`, Vim przekształci znaki tabulacji na spacje. Można skonfigurować liczbę spacji, na które będzie przekształcony jeden znak tabulacji: `:set shiftwidth=4`.

Nie należy jednak pochopnie uaktywniać opcji `expandtab`, trzeba wcześniej wziąć pod uwagę możliwe konsekwencje. W przeciwnym wypadku może czekać niespodzianka, na przykład przy edycji plików Makefile. Cele pośrednie programu make są definiowane jako polecenia wcięte za pomocą znaków tabulacji. Zastąpienie tych znaków spacjami spowoduje zgłoszenie błędu składni. Aby tego uniknąć, należy włączyć autodektekcję typu pliku przez Vim za pomocą `autocmd` i opcję `expandtab` aktywować tylko dla plików Perla:

```
:filetype on
:autocmd FileType $*
perl :set expandtab
```

Aby kontrolować tego typu sytuacje, można posłużyć się poleceniem `:set list`, które powoduje wizualizację w Vimie znaków niedrukowanych. Znak tabulacji jest przedstawiany jako `^I`, a znak końca wiersza jako niebieski

znak '\$'. Polecenie `:set nolist` przywraca domyślny tryb wyświetlania.

Wspomniana wcześniej opcja `shiftwidth` ma jeszcze jedno znaczenie: w połączeniu z opcją `cindent` pozwala zaoszczędzić dużej ilości wpisywania. Po wpisaniu warunku `if ($really) {` i naciśnięciu klawisza `Enter`, Vim wykona wcięcie zgodnie z ustawieniami opcji `shiftwidth` oraz `expandtabs`. Gdy zostanie wpisany znak `}` i naciśnięty `Enter`, Vim automatycznie przesunie zamykający nawias klamrowy w lewo, na początek wiersza. W przypadku niektórych plików to zachowanie edytora nie jest pożądane, należy więc zdefiniować autopolecenie identyfikujące typ pliku i ustawiające tę opcję tylko dla odpowiednich przypadków: `autocmd FileType perl :set cindent`

Czasem trudno zorientować się, że segment kodu powinien być wcięty, zanim nie wpisze się całości. W tym przypadku należy umieścić kursor w nawiasach klamrowych bloku, który ma być wcięty (Rysunek 4a), po czym przejść do trybu poleceń i nacisnąć następującą sekwencję klawiszy: `>i{`, co spowoduje wcięcie zawartości bloku ujętego w nawiasy klamrowe o liczbę spacji określoną opcją `shiftwidth` (Rysunek 4b).

Opcja `:set smarttab` ustawia kolejną funkcję korzystającą z opcji `expandtab`: naciśnięcie klawisza Backspace z kursorem ustawionym przed pierwszą literą wciętego wiersza kodu spowoduje usunięcie znaków wcięcia do początku wiersza, natomiast naciśnięcie klawisza `Tab` przesunie tekst w prawo o odpowiednią liczbę znaków bez używania znaków tabulacji.

Aby przenieść kursor pomiędzy otwierającym a zamykającym blok nawiasem klamrowym, wystarczy ustawić kursor na jednym z tych nawiasów i nacisnąć klawisz `%` w trybie poleceń. Dzięki temu łatwo odszukać brakujące nawiasy klamrowe w przypadku błędów składni Perla.

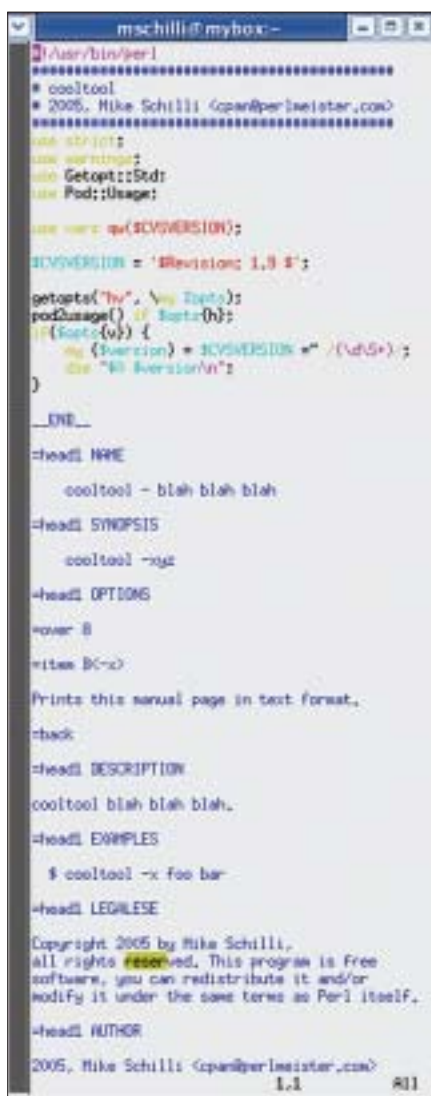
Użytkownicy klawiatur w układzie amerykańskim, chcący wprowadzić znaki spoza tabeli znaków (na przykład znak Å), mogą w trybie wprowadzania zastosować odpowiedni kod, w tym przypadku `Ctrl-K A`. Listę dostępnych kodów znaków niestandardowych można uzyskać wywołując polecenie `:digraphs`.

Listing 1: Polecenia Vim

```
# Wyszukiwanie ciągu 'sub'
/sub
# Rozpoczęcie nagrywania
# makra a
qa
# Wstawienie wiersza przed
znalezionym wierszem,
# Przejście do trybu poleceń
O<ESC>
# Wstawienie 20 znaków '#'
20i#<ESC>
# Skopiowanie wiersza
yy
# Przejście w dół o jeden wiersz
# Wstawienie skopiowanego wiersza
# poniżej bieżącego.
jp
# Wyłączenie nagrywania makra
q
# Wyszukanie kolejnego
wystąpienia ciągu 'sub'
n
# Odtworzenie makra 'a'
@a
# ... powtórzenie.
```

Na dobry początek

Skrypt *tmpl* z [5] zawiera narzędzia przydatne podczas tworzenia zupełnie nowego skryptu w Perlu. Na przykład `$ tmpl -p cooltool` utworzy nowy plik o nazwie *cooltool*. Jak widać na Rysunku 5 skrypt umieszcza w takim pliku kilka wierszy nagłówka, włącza kilka typowych modułów, nieco kodu analizującego opcje wywołania skryptu oraz wy-



```
mschilli@mybox:~$ perl /usr/bin/perl /usr/bin/perl
=====
# cooltool
# 2005, Mike Schilli <cpan@perlmeister.com>
=====
use strict;
use warnings;
use Getopt::Std;
use Pod::Usage;

our $CVSVERSION;

$CVSVERSION = 'Revision: 1.0 $';

getopts("hv", \%opts);
podUsage() if $opts{h};
{
    my ($version) = $CVSVERSION =~ /(v\d+)/;
    my $H_VERSION = "1.0";
}

__END__

=head1 NAME

cooltool - blah blah blah

=head1 SYNOPSIS

cooltool -xyz

=head1 OPTIONS

power 8

writes B<=>

Prints this manual page in text format.

=back

=head1 DESCRIPTION

cooltool blah blah blah.

=head1 EXAMPLES

$ cooltool -x foo bar

=head1 LEGALESE

Copyright 2005 by Mike Schilli.
all rights reserved. This program is free
software, you can redistribute it and/or
modify it under the same terms as Perl itself.

=head1 AUTHOR

2005, Mike Schilli <cpan@perlmeister.com>
1.1 811
```

Rysunek 5: Szkielet nowego skryptu o nazwie *cooltool* utworzony przez skrypt *tmpl*.

świetlanie tekstu pomocy. Skrypt *tmpl* odczytuje parametry swojego działania (jak nazwisko autora tworzonego pliku) z pliku *tmpl* zapisanego w katalogu domowym użytkownika.

Szkielet skryptu *cooltool* już potrafi robić dwie rzeczy: po wywołaniu `$ cooltool -v` zostanie wypisana informacja o wersji, zapisana w zmiennej `$CVSVERSION` automatycz-

nie aktualizowanej przez CVS. Ponadto skrypt potrafi wypisać tekst pomocy (po wywołaniu z opcją `-h`), co zapewnia moduł *Pod::Usage*.

Oczywiście, twórcy skryptów muszą wypełnić brakującą funkcjonalność, lecz dzięki szkieletowi sporo pracy jest już wykonane, dając jednocześnie podwaliny pod dokumentację modułu, co w przypadku każdego skryptu jest rzeczą bardzo cenną.

Uzupełnianie tekstu

Vim zapamiętuje wyrazy wpisane przez użytkownika i uzupełnia wprowadzane teksty po naciśnięciu kombinacji klawiszy `Ctrl-n` w trybie wprowadzania. Jeśli w skrypcie zostanie zdefiniowana zmienna `our $GLOBAL_SUPER_VARIABLE;`, która ma być użyta w innym jego miejscu, nie ma potrzeby ponownego wpisywania całej nazwy. Wystarczy wpisać kilka pierwszych liter, nacisnąć `Ctrl-n` i Vim odczyta nasze myśli.

Jeśli wprowadzone kilka pierwszych znaków nie daje jednoznacznego dopasowania, można nacisnąć `Ctrl-n` kilkakrotnie, aż Vim podsunie właściwą podpowiedź, `Ctrl-p` umożliwia przewijanie podpowiedzi wstecz. Ta funkcja pozwala zaoszczędzić mnóstwo czasu i klawiszologii.

Tags

Programiści języka C z pewnością znają program *ctags*, który tworzy tzw. plik tagów dla edytora Vim. Po wczytaniu tego pliku programista może ustawić kursor w obrębie wywołania funkcji i nacisnąć kombinację `CTRL-J` w trybie poleceń, co spowoduje przeniesienie kursora do definicji funkcji, niezależnie od tego w jakim pliku ta definicja się znajduje.

Aby zmusić Vim'a do przejścia do pliku źródłowego modułu `LWP::UserAgent` języka Perl, ustawiając kursor pośród liter `LWP::UserAgent`, należy wykonać dwie operacje. Po pierwsze, należy poinformować Vim, że słowa kluczowe w Perlu mogą zawierać dwukropek. W tym celu należy wywołać polecenie `:set iskeyword+ =.`. Po drugie, Vim potrzebuje pliku „tagów”, czyli indeksu zainstalowanych pakietów, który wskazuje się edytorowi za pomocą polecenia typu `:set tags=/home/mschilli/.ptags.txt`.

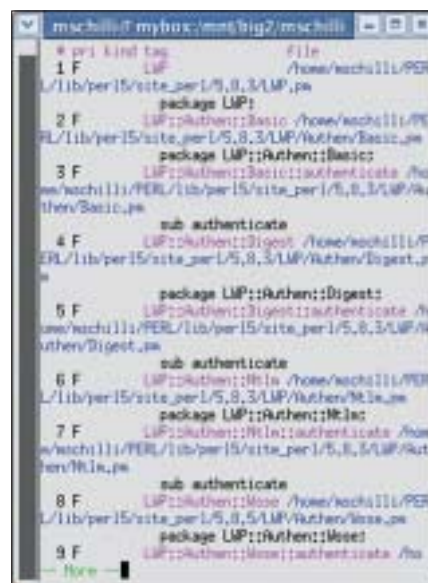
Po ustawieniu kursora na nazwie modułu i naciśnięciu kombinacji `CTRL-J` edytor otworzy kod źródłowy modułu. Alternatywnie można podać nazwę modułu jako parametr wiersza poleceń, na przykład `:tag LWP::UserAgent`.

Naciśnięcie kombinacji `CTRL-T` podczas przeglądania modułu spowoduje powrót do punktu wyjścia. Magia funkcjonowania tego mechanizmu jest zasługą pliku *.ptags.txt*, który jest tworzony przez Listing 2.

Mimo że większość osób korzysta z graficznego interfejsu użytkownika, zdarza się potrzeba jednoczesnego wyświetlenia dwóch plików w jednym oknie, a należy przyjąć, że większość użytkowników edytora Vim niechętnie sięga po myszkę, gdy mają ręce zajęte wpisywaniem tekstu. Jeśli zamiast kombinacji `Ctrl-J` zastosuje się `Ctrl-W-J` z kursorem ustawionym na słowie klu-



Rysunek 6: Vim w trybie podzielonego okna.



Rysunek 7: Zastosowanie wyrażeń regularnych do wyszukiwania tagów. Wpisanie `/^LWP` wyświetli menu z ponumerowanymi opcjami.

czowym, okno edytora zostanie podzielone na dwie części. W dolnej części pozostanie kod edytowanego właśnie pliku, w górnym pojawi się moduł, do którego się odwołał. Kombinacja `Ctrl-WW` służy do przełączania pomiędzy panelami. Wywołanie polecenia `:quit` w górnym oknie zamyka to okno, przywracając wyświetlanie edytowa-

dobry” analizator kodu Perla – w rzeczywistości analizator ten jest zdumiewająco dobry. Moduł *PPI* dostępny w archiwach CPAN zawiera *PPI::Document*. Jego metoda *load()* odczytuje moduł Perla, dzieli go na leksemy, które przechowuje jako węzły w strukturze drzewiastej.

Skrypt *ppitags* wykorzystuje *File::Find* do analizy katalogów z globalnej tablicy *@INC*. Dla każdej odszukanej pozycji *File::Find* uruchamia funkcję *file_wanted*. Jeśli pozycja jest katalogiem, a nie plikiem, wiersz 34 aktualizuje tablicę skojarzeniową *%dirs* sprawdzając, czy ścieżka już była przetwarzana. Jeśli tak, wiersz 33 ustawia zmienną *\$File::Find::prune* na wartość 1, która wskazuje *File::Find*, że może pominąć resztę katalogu i jego podkatalogów. Wiersz 37 powoduje, że skrypt ignoruje wszystkie znalezione pliki oprócz tych, których nazwy mają rozszerzenie *.pm* (moduły Perla).

Wiersz 40 przetwarza bieżący moduł Perla. Błędy są obsługiwane w wierszu 43 (*PPI* nie jest jeszcze doskonały), wypisywane są ostrzeżenia i problematyczny moduł jest pomijany.

Po przeanalizowaniu modułu w wierszu 51 wywoływana jest metoda *find()* obiektu *PPI::Document*, która przechodzi po kolei

przez wszystkie leksemy źródła modułu, wywołując dla każdego napotkanego leksemu funkcję *document_wanted*.

Funkcja sprawdza, czy leksem jest typu *PPI::Statement::Package* lub *PPI::Statement::Sub*, to znaczy jest definicją *package* lub *sub* w kodzie Perla.

Definicja *package* oznacza, że jest zdefiniowana w wierszu typu *package LWP::UserAgent*; co z kolei oznacza, że w *PPI* zostaną wyodrębnione cztery leksemy: *package*, spacja, nazwa modułu oraz zamykający średnik. Dla *ppitags* interesująca jest tylko nazwa modułu, to znaczy trzeci potomek węzła, który był przekazany do funkcji *document_wanted()* jako *\$_[1]*. Metoda *child()* odszukuje łańcuch znaków „*LWP::UserAgent*”: *\$_[1]->child(2)*.

Wiersz 55 wyszukuje definicje typu *sub func {}* i wydobywa nazwy funkcji i metod, aby umożliwić mechanizmowi tagów identyfikowanie konstrukcji typu *LWP::Debug::trace* i odszukanie miejsca w module *LWP::Debug*, w którym występuje definicja funkcji *trace*.

Gdy przetwarzana jest definicja *package*, *ppitags* zapisuje nazwę pakietu jako nazwę bieżącą, którą następnie dopisuje jako przedrostek nazw funkcji znalezionych w pakiecie. Ten mechanizm nie zadziała w przypadku definicji pakietów zagnieżdżonych w blokach, lecz sprawdza się w 99.9% przypadków.

Instrukcja *push* w wierszu 80 zapisuje na końcu tablicy *<@>found* ciąg znaków złożony z wymaganego taga (nazwa pakietu lub pełna nazwa funkcji), absolutnej ścieżki źródła pakietu oraz wyrażenia regularnego, które posłuży do wyszukania pakietu lub funkcji w pliku źródłowym. Funkcja zdefiniowana w wierszu 91, *regex_from_node*, składa wyrażenie regularne ze wszystkich znaków w wierszu od początku aż do poszukiwanego leksemu. W przypadku podprocedur *\$node->content()* zwraca nagłówki funkcji wraz z jej treścią. Z tego powodu w wierszu 98 usuwane są wszystkie wiersze oprócz pierwszego, natomiast wiersze od 100 do 109 odczytują leksemy od końca aż do napotkania początku wiersza. Na końcu petli *while* zmienna *\$regex* zawiera wiersz źródła od początku wiersza do poszukiwanego leksemu. Wiersz 114 wykorzystuje tę zmienną do utworzenia wyrażenia regularnego typu */^.../* (z uchwycem początku wiersza). Instrukcja wyszukiwania z zastępowaniem z wiersza 112 służy wyczyszczeniu z kodu Perla znaków specjal-

nych, które mogą zakłócić wyrażenie regularne, poprzedzając je znakami odwróconego ukośnika.

Skrypt *ppitags* tworzy plik *~/.ptags.txt* zawierający trójkolumnową listę wpisów w formacie: *Pakiet/Podprocedura [tab] nazwa_pliku [tab] wyrażenie_regularne*. Ten plik Vim wczyta po wywołaniu polecenia *:set tags=* jak to zostało opisane wcześniej, dzięki czemu programista będzie miał możliwość przechodzenia pomiędzy edytowanym kodem a kodem modułu.

Wydaje się sensowne, aby uruchamiać *ppitags* raz dziennie w ramach mechanizmu cron, aby plik *~/.ptags.txt* był zawsze aktualny. Można również rozszerzyć skrypt, pozwalając Vimowi identyfikować w pełni kwalifikowane zmienne typu *our* (na przykład *\$Text::Wrap::columns*) i odszukiwać miejsca ich definicji w plikach źródłowych modułów.

Trwały zapis

Przy uruchamianiu Vim odczytuje plik *.vimrc* zapisany w katalogu użytkownika, co daje możliwość zmuszenia Vima do wykonania sekwencji działań jeszcze zanim załaduje plik przeznaczony do edycji. Po uruchomieniu programu zdarza się modyfikować opcje edytora i często chcemy, aby zmiany wprowadzone w trybie interaktywnym zostały zapamiętane na potrzeby następnych sesji. Zamiast przepisywać te polecenia do pliku *.vimrc*, można po prostu wywołać polecenie *:mkvimrc*, które zapisze aktualne ustawienia w pliku *~/.vimrc*.

Pod adresem [1] można znaleźć przykładową konfigurację, zawierającą ustawienia omówione w tym artykule. Pamiętajmy, że oszczędność czasu i nadgarstków daje więcej korzyści, niż może się wydawać! ■

Listing 3: .vimrc

```
version 6.0
:map !L iuse Log::Log4perl >
qw(:easy);<RETURN>Log::Log4perl->>
easy_init($DEBUG);<RETURN><ESC>
:map F o<ESC>43i#<ESC>yosub >
{<ENTER><ESC>Pk$i
map f !Gperl -MText::Autoformat >
-e'autoformat{right=>70}'^V^M
set backspace=2
set fileencodings=utf-8,latin1
set formatoptions=tcql
set helplang=en
set history=50
set hlsearch
set ruler
set shiftwidth=4
:autocmd FileType perl :set cindent
:autocmd FileType perl :set
expandtab
set smarttab
:nnoremap X :w<Enter>:!perl -c >
%<Enter>
:set tags=/home/mschilli/.ptags.txt
:set iskeyword+=:
```

INFO

- [1] Listingi z tego artykułu: <ftp://www.linux-magazine.com/Magazine/Downloads/57/Perl>
- [2] Strona projektu Vim: <http://www.vim.org>
- [3] Steve Oualline, „vi Improved – Vim”, New Riders, 2001
- [4] Mike's Script Archive: <http://perlmeister.com/scripts>
- [5] Skrypt *tmpl*: <http://perlmeister.com/scripts/tmpl>