

Przeszukiwanie środowiska pracy w Perlu

# SZUKAJ!

Prezentujemy skrypt Perla, który tworzy bazę MySQL umożliwiającą znajdowanie plików w ułamku sekundy.

MICHAEL SCHILLI

**G**dzie podział się mój napisany wczoraj skrypt? Gdzie są najnowsze pliki, które zajmują większość miejsca na dysku albo pliki, których nie oglądano od co najmniej trzech lat? I gdzie wcięło plik tekstowy z ostatniego tygodnia, zawierający słowa „Michael” i „podróż”?

Oczywiście, nic nie stoi na przeszkodzie, żeby przeszukać dysk plik po pliku i znaleźć daną informację. W ciągu ostatnich lat wprowadzenie taniach i jednocześnie ogromnych dysków twardej spowodowało, że użytkownicy nie przejmują się już porządkowaniem swoich katalogów domowych. Narzędzia takie jak *find* muszą często przeglądać dziesiątki lub setki nieistotnych plików, zanim zwrócą pożądane wyniki. Zajmuje to czas, a nadmiar czasu jest luksusem, na który niewielu może sobie pozwolić.

Narzędzia takie jak *slocate* przeszukują drzewa katalogów w nocy, aby następnego dnia pomagać użytkownikom szybko znajdować pliki według ich nazw. Google Desktop [2] i Spotlight dla MacOS X idą o krok dalej,

tworząc meta-indeks i pozwalając użytkownikom szukać plików spełniających zadane kryteria.

Dzisiaj przedstawimy skrypt Perla *rummage*, umożliwiający wyszukiwanie zasobów na dysku. Bierze on pod uwagę nie tylko nazwy plików, ale zapamiętuje, kiedy pliki pojawiły się po raz pierwszy i kiedy zostały zmienione po raz ostatni. Do tworzonej bazy MySQL (Rysunek 1) dodaje rozmaite fragmenty meta-informacji dotyczących poszczególnych plików, a dla plików tekstowych tworzy indeks pełnotekstowy, umożliwiający przeszukiwanie treści przy użyciu słów kluczowych.

## Przeszukiwanie pełnotekstowe

W wersji 3.23.23 bazy MySQL wprowadzono opcję FULLTEXT, używaną do oznaczania kolumn w tabelach, dla których ma być przeprowadzane przeszukiwanie całej ich zawartości. W wersji 4.0.1 dodano operatory logiczne dla wyszukiwanych słów kluczowych. Możliwe jest nawet tworzenie tak zwanych

list wykluczających (stop lists), które zabraniają indeksowania powszechnych, ale bezużytecznych słów. Baza danych umożliwia nawet rozszerzanie zapytań (query expansion), zwracające listę dokumentów, które zawierają słowa znajdujące się w innych dokumentach spełniających warunki zapytania. Szybkość realizowania takich zapytań pozostawia jednak wiele do życzenia. Jeśli zaś każdy dokument tekstowy zostanie zapisany w bazie danych, korzystanie z takiej bazy szybko staje się niewygodne i nieefektywne.

Moduł Perla *DBIx::FullTextSearch* używający mechanizmów MySQL-a ma również parę wad. Indeksowanie jest wolnym procesem, a staje się jeszcze wolniejsze, kiedy za-indeksowanych jest ponad 30 000 plików.

Dlatego też *rummage* używa sprawdzonego programu SWISH-E, który indeksuje i przeszukuje ze zdumiewającą szybkością. Umożliwia on przeszukiwanie za pomocą słów kluczowych, jak i wyszukiwanie zadanych fraz, a także bardzo dobrze się skaluje. Korzystanie ze SWISH-E ułatwia moduł

SWISH::API::Common z archiwum CPAN, koncentrujący się na najczęściej używanych funkcjach. Należy jednak wspomnieć, że SWISH-E nie potrafi usuwać plików z już utworzonego indeksu, co – dla zachowania aktualności danych – wymusza codzienne

Polecenie `rummage -n 20` pozwala znaleźć 20 ostatnio zmodyfikowanych plików. Jeśli nie podasz liczby, polecenie przyjmie wartość domyślną i wyświetli 10 ostatnio zmienionych plików. Polecenie `rummage -m „7 day”` zwróci wszystkie pliki zmodyfikowane

czasu ostatniej aktualizacji bazy, czyli najczęściej do poprzedniej nocy. Program `rummage` po prostu nie uwzględnia niczego, co wydarzyło się później.

Przypuszczalnie będzie trzeba zmodyfikować pierwszą część programu `rummage`, aby dopasować ją do danego środowiska pracy. Stała `$MAX_SIZE` definiuje maksymalną długość zawartości, która ma być zaindeksowana z pojedynczego pliku tekstowego. Jeśli operator `-T` Perla wywoływany w `SWISH::API::Common` zidentyfikuje 100-megabajtowy plik dziennika jako plik tekstowy, prawdopodobnie nie chcesz indeksować całego pliku. Wartość `100_000` oznacza, że zaindeksowane zostanie jedynie jego pierwsze 100 kilobajtów.

W następnym wierszu zmienna `$DSN`, definiująca nazwę źródła danych (Data Source Name) dla modułu DBI-Class, określa sterownik dla bazy danych (w przypadku MySQL-a jest to `DBD::mysql`) oraz nazwę bazy (`mts`). Natomiast `@DIRS` jest tablicą nazw katalogów, które są rekurencyjnie przeszukiwane przez program `rummage`. W wierszu 24 ewentualne dowiązania symboliczne zamieniane są na wskazywane przez nie katalogi. Jeśli indeksowanie całego katalogu domowego zabiera zbyt dużo czasu, można ograniczyć indeksowanie do jednego lub wielu podkatalogów, takich jak lokalna wersja robocza CVS.

Wiersz 27 zawiera deklarację funkcji `search`, która wypisuje rezultaty różnych zapytań. Deklaracja prototypu funkcji jest konieczna, żeby określić, że `search` jako swojego jedyne argumentu oczekuje wartości skalarnej. Sprawia to, że metody klasy `DBI::Class`, `search()` i `search_like()`, zostaną wywołane w kontekście skalarnym, zwraca-

Field	Type	Null	Key	Default	Extra
fileid	int(11)		PRI	NULL	auto_increment
path	varchar(255)	YES	MUL	NULL	
size	int(11)	YES		NULL	
atime	datetime	YES		NULL	
mtime	datetime	YES		NULL	
first_seen	datetime	YES		NULL	
type	varchar(255)	YES		NULL	
checked	int(11)	YES		NULL	

8 rows in set (0.00 sec)

Rysunek 1: Schemat tabeli 'file', w której 'rummage' przechowuje meta-dane plików.

powtórne indeksowanie plików. Wykonywane każdej nocy zadanie `crona` może z łatwością obsłużyć setki tysięcy plików, co zupełnie wystarcza normalnym użytkownikom.

## Metody

Po pierwszej sesji indeksowania poprzez polecenie `rummage -u` meta-dane i indeks pełnotekstowy stają się dostępne dla użytkowników. Polecenie `rummage -k query` znajduje pliki zawierające podane słowo kluczowe. W Ramce 1 podajemy parę przykładów przeszukiwania za pomocą słów kluczowych i zapytań o różne meta-dane.

Jak pokazano na schemacie na Rysunku 1, baza MySQL-a przechowuje pełną ścieżkę dostępu do każdego pliku, jego rozmiar w bajtach, czas pierwszego pojawienia się w systemie plików, czas ostatniego dostępu i ostatniej modyfikacji.

Plik o nazwie `call.sgml`, ukryty gdzieś w mrocznych otchłaniach zaindeksowanej hierarchii plików, znaleźć można przez wywołanie `rummage -p call.sgml`. Mechanizm działania `rummage` polega na zamianie `call.sgml` na wzorzec SQL-a `%call.sgml%` i przeszukaniu tabeli `file` zapytaniem `WHERE path LIKE „%call.sgml%”`. Zapytania o ścieżki względne, takie jak `examples/call.sgml`, także będą działały, ale w takim przypadku `rummage` znajdzie zadany plik tylko wtedy, jeśli znajduje się on w podkatalogu `examples`.

w ciągu ostatniego tygodnia. W tym celu wywoła ono następujące zapytanie MySQL-a:

```
SELECT * FROM file
WHERE DATE_SUB(NOW(),
INTERVAL 7 DAY) <= mtime
```

każąc obliczyć bazie MySQL, czy data modyfikacji dla każdego rekordu wypadła wcześniej niż jeden tydzień wstecz. W razie potrzeby możesz zastąpić liczbę dni wyrażeniem typu `3 month` (3 miesiące) lub `18 hour` (18 godzin). Oczywiście, żadne z tych wyrażen nie odnosi się do chwili obecnej, lecz do

## Opcje Rummage

```
rummage -u -v # Odśwież lub utwórz bazę danych;
               # -v włącza szczegółowe rejestrowanie
               # w pliku dziennika
rummage -k 'linux' # Szukaj słowa kluczowego "linux"
rummage -k "mike schilli" # Szukaj frazy "mike schilli"
rummage -k 'foo AND (bar OR baz)' # Szukaj dokumentów zawierających "foo" i "bar"
                                   # lub "foo" i "baz"
rummage -k 'torvald*' # Szukaj przy użyciu symboli wieloznacznych
rummage -p pathlike # Szukaj według nazwy lub ścieżki dostępu pliku
rummage -n 20 # Wyświetl 20 ostatnio modyfikowanych plików
rummage -m '7 day' # Wyświetl wszystkie pliki zmodyfikowane w ciągu ostatniego tygodnia
```

jąc iterator, który może być wykorzystany przez funkcję *psearch*.

Bez prototypu metoda *search()* w wyrażeniu *psearch(\$db->search(...))* wywołana byłaby w kontekście listowym i zwracała listę trafień zamiast iteratora.

Funkcja *getopts()* analizuje przekazywane parametry wiersza poleceń. Jeśli parametr wymusza aktualizację bazy danych (*-u*), wiersz 32 włącza mechanizm Log4perl. Jeśli użytkownik włączył opcję szczegółowego rejestrowania (*-v*), poziom rejestrowania ustawiany jest na \$DEBUG. Domyślną wartością poziomu rejestrowania jest \$INFO, która sprawia, że w pliku dziennika zachowywane są jedynie komunikaty informacyjne.

Plik dziennika jest nadpisywany za każdym razem, żeby uniknąć zapełnienia dysku twardego. Innym rozwiązaniem byłoby skonfigurowanie mechanizmu Log4perl przy użyciu *Log::Dispatch::FileRotate*.

W wierszu 41 wywoływana jest funkcja *db\_init()* zdefiniowana w wierszu 186; funkcja ta tworzy tabelę *file* w bazie danych, jeśli taka jeszcze nie istnieje. Dodatkowo definiuje ona indeks dla kolumny *path*, żeby umożliwić programowi *rummage* późniejsze szybkie sprawdzanie, czy rekord dla danego pliku już istnieje i czy data ostatniej modyfikacji pliku została zmieniona. Wszystko to oznacza, że pierwsza indeksacja przeprowadzana przez *rummage* może trochę potrwać. Z drugiej strony, późniejsze aktualizacje odbywać się będą znacznie szybciej.

W wierszu 44 moduł *Class::DBI::Loader* łączy się z bazą danych, generując obiektową reprezentację bazy danych dla potrzeb modułu *Class::DBI*. Następnie tworzona jest obiektowa reprezentacja tabeli *file*, będąca instancją klasy *Rummage::File*. Jeśli jakieś wywołanie metody *search()* zwraca iterator, jest on przetwarzany przez funkcję *psearch()*, która po prostu wywołuje metodę *->next()* iteratora, dopóki zwraca ona trafienia. Dla każdego trafienia wywoływana jest jego metoda *path()*, zwracająca ścieżkę do pliku, oraz metoda *mtime()*, zwracająca datę ostatniej modyfikacji.

Nie wszystkie zapytania można łatwo wykonać przy użyciu wysokopoziomowych funkcji *Class::DBI*. Kiedy zapytania stają się bardziej skomplikowane, używając *Class::DBI* można „zniżyć się” do poziomu SQL-a. Metoda *set\_sql* pozwala na definiowanie zapytań, takich jak *newest* w wierszu 92, którego można potem użyć jako wysokopoziomowej metody klasy *Class::DBI* o nazwie *search\_newest*.

```

mysql> SELECT path, size FROM file
mysql> ORDER BY size DESC LIMIT 10;
+-----+-----+
path,size
+-----+-----+
'/mnt/big2/wschilli11.do.not.delete/BAC00PS/cv050529.tgz', '659375002'
'/mnt/big2/wschilli11.do.not.delete/.swish-cowse/default.idx', '568302711'
'/mnt/big2/wschilli11.do.not.delete/IM5TML/bloomba050119.zip', '349236187'
'/mnt/big2/wschilli11.do.not.delete/mail/up', '282842236'
'/mnt/big2/wschilli11.do.not.delete/rescue/h.img', '204800000'
'/mnt/big2/wschilli11.do.not.delete/D70-book/cdrom.iso', '113631232'
'/mnt/big2/wschilli11.do.not.delete/DEV/private/rundbrief/55/angelika.ps', '103718914'
'/mnt/big2/wschilli11.do.not.delete/rescue/hdd1.img', '102400000'
'/mnt/big2/wschilli11.do.not.delete/rescue/hdd1_02.img', '102400000'
'/mnt/big2/wschilli11.do.not.delete/rescue/hdd1_01.img', '102395004'
+-----+-----+
110 rows of 2 fields returned
mysql>

```

Rysunek 2: Używanie zapytań MySQL-a do lokalizowania największych plików.

## Zawsze na czasie

Kiedy przekazemy programowi *rummage* parametr wiersza poleceń *-u*, będzie on przeszukiwał system plików używając modułu *File::Find* i doda najświeższe meta-informacje do bazy danych. Na początku wywołanie *UPDATE* zdefiniowane w wierszu 107 i uruchamiane w wierszu 83, zeruje wartości kolumny *checked* dla wszystkich rekordów w tabeli. Jeśli funkcja przeszukująca znajdzie dany plik w systemie plików, odpowiedni rekord jest oznaczony jako sprawdzony przez ustawienie wartości kolumny *checked* na 1. Plik odpowiadający rekordowi, który po zakończeniu przeszukiwania nadal ma wartość *checked=0*, musiał zniknąć z systemu od momentu ostatniej aktualizacji. Taki wpis musi być usunięty z bazy danych i z indeksu przeszukiwania pełnotekstowego.

Wiersz 115 uruchamia funkcję *find*, która zaczyna szukać w podanych katalogach i zagłębia się w hierarchię plików. Funkcja *wanted* zdefiniowana w wierszu 140 wywoływana jest za każdym razem, kiedy znajdowany jest nowy element tej hierarchii. Wiersz 142 błyskawicznie odrzuca wszystkie elementy, które nie wyglądają jak pliki. Polecenie *stat* w wierszu 150 zwraca rozmiar pliku w bajtach oraz czas jego ostatniego odczytu i zapisu.

Jeśli znaleziony plik ma swój wpis w bazie danych, wiersz 160 sprawdza, czy data ostatniej modyfikacji pliku jest identyczna z wartością przechowywaną w bazie danych. W przeciwnym przypadku wiersze 165 do 167 uaktualniają meta-informacje (*mtime* – czas ostatniej aktualizacji, *atime* – czas ostatniego dostępu i *size* – rozmiar). Jeśli plik nie ma jeszcze swojego wpisu w bazie danych,

metoda *create* w wierszu 170 tworzy nowy rekord. Wywołanie metody *checked()* w wierszu 180 przypisuje polu *checked* wartość 1, a następujące po nim wywołanie *update()* zamyka transakcję.

## Konwersja formatu czasu

MySQL oczekuje, że pola typu DATETIME podawane będą w formacie „YYYY-MM-DD HH:MM:SS”, ale funkcja *stat* Perla zwraca czas uniksowy w sekundach. Na szczęście moduł *Time::Piece::MySQL* z archiwum CPAN udostępnia metodę *mysql\_datetime*. Funkcja *mysqltime*, zdefiniowana w wierszu 238 programu *rummage*, skraca to wywołanie.

Innym rozwiązaniem byłoby użycie wewnętrznej funkcji MySQL-a *FROM\_UNIXTIME()*, ale *rummage* musiałby w tym celu przebiegać się przez warstwę abstrakcji dostarczoną przez moduł *DBI::Class*.

## Śmieci i pożeracze miejsca na dysku

Program *rummage* można łatwo rozbudować. Przed dodaniem do *rummage* inteligencji, można – używając programu klienta *mysql* – spróbować pobawić się meta-danymi plików, które *rummage* już przetworzył zapytaniami *DBI::Class*.

Powłoka *DBI dbish* z archiwum CPAN również udostępnia użyteczne funkcje. Pozwala na łączenie się z bazą, która jest obsługiwana przez *DBI*, i wykonywanie zapytań SQL. Powłoka ta instalowana jest razem z modułem *DBI::Shell* z archiwum CPAN. Z bazą MySQL łączy nas wywołanie *dbish*

## Listing 1: rummage

```

001 #!/usr/bin/perl -w
002 #####
003 # rummage - Indeksowanie i
przeszukiwanie
004 # katalogu domowego
005 # Mike Schilli, 2005
006 # <m@perlmeister.com>
007 #####
008 use strict;
009
010 use Getopt::Std;
011 use File::Find;
012 use DBI;
013 use Class::DBI::Loader;
014 use Log::Log4perl qw(:easy);
015 use SWISH::API::Common;
016 use Time::Piece::MySQL;
017
018 my $MAX_SIZE = 100_000;
019 my $DSN = "dbi:mysql:dts";
020 my @DIRS = ("${ENV{HOME}}");
021 my $COUNTER = 0;
022
023 @DIRS = map {
024     -l $_ ? readlink $_ : $_
025 } @DIRS;
026
027 sub psearch($);
028 getopts( "un:m:k:p:v",
029     \my %opts );
030
031 if ( $opts{u} ) {
032     Log::Log4perl->easy_init( {
033         level =>
034             $opts{v} ? $DEBUG :
035                 $INFO,
036         file =>
037             ">/tmp/rummage.log",
038     });
039 }
040
041 db_init($DSN);
042
043 my $loader =
044     Class::DBI::Loader->new(
045         dsn => $DSN,
046         user => "root",
047         namespace => "Rummage",
048     );
049
050 my $filedb =
051     $loader->find_class("file");
052
053 my $swish =
054     SWISH::API::Common->new(
055         file_len_max => $MAX_SIZE,
056         atime_preserve => 1,
057     );
058
059 # Szukanie słów kluczowych
060 if ( $opts{k} ) {
061     my @docs = $swish->search(
062         $opts{k} );
063     print $_->path(), "\n"
064         for @docs;
065
066 # Szukanie według czasu
modyfikacji (mtime)
067 } elsif ( $opts{m} ) {
068     $filedb->set_sql(
069         modified => qq{
070             SELECT __ESSENTIAL__
071             FROM __TABLE__
072             WHERE DATE_SUB(NOW(),
073                 INTERVAL $opts{m}) <= mtime
074             });
075     psearch(
076         $filedb->search_modified()
077     );
078
079 # Szukanie według ścieżki
dostępu
080 } elsif ( $opts{p} ) {
081     psearch(
082         $filedb->search_like(
083             path => "%$opts{p}%"
084         )
085     );
086
087 # Przeszukiwanie najnowszych
088 } elsif ( exists $opts{n} ) {
089     $opts{n} = 10
090     unless $opts{n};
091
092     $filedb->set_sql(
093         newest => qq{
094             SELECT __ESSENTIAL__
095             FROM __TABLE__
096             ORDER BY mtime DESC
097             LIMIT $opts{n}
098         });
099     psearch(
100         $filedb->search_newest()
101     );
102
103
104 # Indeksowanie katalogu domowego
105 } elsif ( $opts{u} ) {
106     # Wzyceruj całą kolumnę
"checked"
107     $filedb->set_sql(
108         "unchecked_all", qq{
109             UPDATE __TABLE__
110             SET checked=0
111         });
112     $filedb->sql_uncheck_all()
113     ->execute();
114
115     find( \&wanted, @DIRS );
116
117     # Uaktualnij indeks słów
kluczowych
118     $swish->index_remove();
119     $swish->index(@DIRS);
120
121     # Usuń wszystkie "martwe"
wpisy
122     # w bazie danych
123     $filedb->set_sql(
124         "delete_dead", qq{
125             DELETE FROM __TABLE__
126             WHERE checked=0
127         });
128     $filedb->sql_delete_dead()
129     ->execute();
130
131     } else {
132         LOGDIE "opcje: $0 [-u] ",
133             "[-v] [-n [N]] ",
134             "[-p pathlike] ",
135             "[-k keyword] ",
136             "[-m interval]";
137     }
138
139 #####
140 sub wanted {
141     #####
142     return unless -f;
143
144     my $fn = $File::Find::name;
145
146     DEBUG ++$COUNTER, " $fn";
147
148     my ( $size, $atime,
149         $mtime ) =
150         ( stat($_) )[ 7, 8, 9 ];
151     $atime = mysqltime($atime);
152     $mtime = mysqltime($mtime);
153
154     my $entry;
155
156     if ( ($entry) =
157         $filedb->search(
158             path => $fn) ) {
159

```

## Listing 1: rummage

```

160 if ( $entry->mtime() eq
161     $mtime ) {
162     DEBUG "$fn niezmieniony";
163 } else {
164     INFO "$fn zmieniony";
165     $entry->mtime($mtime);
166     $entry->size($size);
167     $entry->atime($atime);
168 }
169 } else {
170     $entry = $filedb->create(
171         { path      => $fn,
172           mtime     => $mtime,
173           atime     => $atime,
174           size      => $size,
175           first_seen =>
176             mysqltime(time()),
177         });
178 }
179
180 $entry->checked(1);
181 $entry->update();
182 return;
183 }
184
185 #####
186 sub db_init {
187     #####
188     my ($dsn) = @_;
```

```

189
190 my $dbh =
191     DBI->connect( $dsn,
192                 "root", "",
193                 { PrintError => 0 } );
194
195 LOGDIE "połączenie nie
196 powiodło sie: ",
197     DBI::errstr unless $dbh;
198
199 if ( !$dbh->do(
200     q{select * from
201         file limit 1}
202     )) {
203     $dbh->do( q{
204         CREATE TABLE file (
205             fileid      INTEGER
206                 PRIMARY KEY
207                 AUTO_INCREMENT,
208             path        VARCHAR(255),
209             size        INTEGER,
210             mtime       DATETIME,
211             atime       DATETIME,
212             first_seen  DATETIME,
213             type        VARCHAR(255),
214             checked     INTEGER
215         )) or LOGDIE
216         "Nie można utworzyć
217         tabeli";
```

```

216
217     $dbh->do( q{
218         CREATE INDEX file_idx
219             ON file (path)
220     });
221 }
222 }
223
224 #####
225 sub psearch($) {
226     #####
227     my ($it) = @_;
228
229     while ( my $doc =
230         $it->next() ) {
231         print $doc->path(), " (",
232             $doc->mtime(), ")",
233             "\n";
234     }
235 }
236
237 #####
238 sub mysqltime {
239     #####
240     my ($time) = @_;
241     return Time::Piece->new(
242         $time->mysql_datetime());
243 }
```

*dbi:mysql:<TABLE> user password*. Rysunek 2 pokazuje powłokę w akcji: zapytanie SQL-a o dziesięciu największych zjadaczy przestrzeni dyskowej, takie jak

```
SELECT path, size FROM file
ORDER BY size DESC LIMIT 10;
```

szybko ujawni winnych i zmusi ich do błagania o litość.

Poniższe polecenie SQL-a znajduje dziesięć najstarszych plików, których nikt nie czytał od lat:

```
SELECT path, atime FROM file
ORDER BY atime ASC LIMIT 10;
```

Pliki tekstowe są codziennie przetwarzane przez program indeksujący. W związku z tym – jeśli tylko system plików nie został podłączony z opcją *noatime* – ostatnie daty dostępu do plików nie będą nigdy starsze niż jeden dzień. Dlatego też konstruktor *SWISH::API::Common->new()* ustawia pa-

rametr *atime\_preserve*, zmuszając moduł do „oszukiwania”: przywracania czasu ostatniego dostępu dla każdego indeksowanego pliku.

## Instalacja

Za pomocą powłoki CPAN instalacja wymaganych modułów Perla powinna udać się bez żadnego kłopotu. Narzędzie *mysqladmin* pomoże utworzyć bazę *fts* w MySQL-u: *mysqladmin -user=root create fts*. Program *rummage* zadba automatycznie o tabele w tej bazie. Następujący wpis w pliku *crontab* wywoła *rummage* raz dziennie o 3:05.05 03 \*\*\* *LD\_LIBRARY\_PATH=/usr/local/lib/home/user/bin/rummage -u -v >/dev/null 2>&1*

Baza danych MySQL jest częścią składową większości współczesnych dystrybucji Linuksa. Jeśli jej nie posiadasz, możesz za darmo ściągnąć aktualną stabilną wersję 4.1 ze strony [mysql.com](http://www.mysql.com).

Program indeksujący *swish-e* i moduł *SWISH::API* dostępne są na stronie [swish-e.org](http://swish-e.org). Moduł *SWISH::API::Common* z archi-

wum CPAN spróbuje zainstalować oba z nich automatycznie. Jeśli się mu nie powiedzie, możesz ściągnąć program *swish-e 2.4.3* lub nowszy i uruchomić *./configure; make install*, aby go zainstalować. Moduł *SWISH::API* jest zawarty w tej dystrybucji. Następujące polecenia

```
cd perl
LD_RUN_PATH=$$
/usr/local/lib perl Makefile.PL
make install
```

przeprowadzą instalację. ■

## INFO

[1] Listingi z tego artykułu: <http://www.linux-magazine.com/Magazine/Downloads/59/Perl>

[2] Google Desktop Search: <http://desktop.google.com>