Desktop search in Perl

GO GET IT!

www.sxc.hu

On a big, busy Linux desktop, it is too easy for files to get lost. We'll show you a Perl script that creates a MySQL database to find files in next to no time.

BY MICHAEL SCHILLI

Now where did I store that script I put together yesterday? Which are the newest files, which take up the most disk space, or which files have not been touched for at least three years? And where was that text file that I wrote last week containing the words "Michael" and "raise?"

Of course, there is nothing to stop you navigating the disk level by level and retrieving the information you need. Cheap but enormous hard disks have led to users no longer bothering to tidy up their home directories in recent years; *find* and other utilities often need to navigate tens or even hundreds of thousands of irrelevant entries before they come up with the goods. That takes time, and time is a luxury that many people don't have.

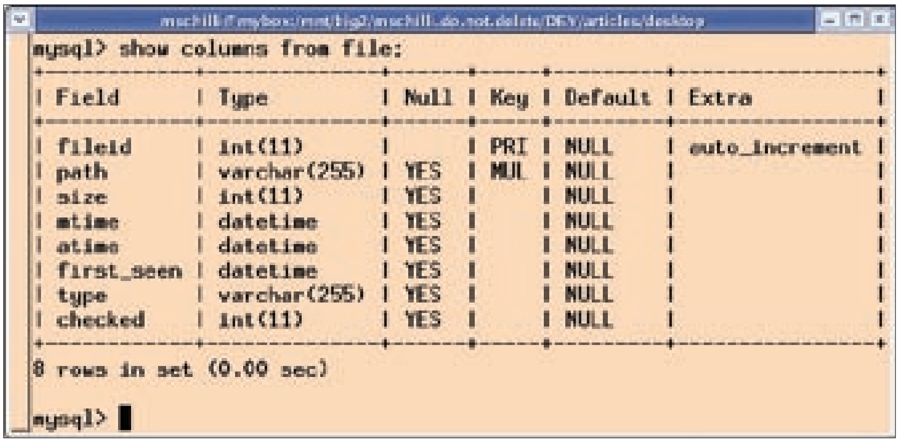
Utilities such as *slocate* climb around the filesystem tree at night helping users to quickly find files by path the next day. The Google desktop [2] and Spotlight on MacOS X take this one step further, by creating a meta-index and helping users to discover files based on a variety of properties.

The script we will be looking at today, *rummage*, implements a Perl-based desktop search. It not only takes filenames into consideration, but also remembers when files first appeared, and when they were last changed. It adds various snippets of meta-information for each file to a MySQL database (Figure 1) and creates a full-text index for text files, allowing users to browse their content later using a keyword-based search.

Full-Text to the Max

Version 3.23.23 of MySQL introduced a FULLTEXT option, which can be used to tag columns in tables and perform full-text searches against the content later. 4.0.1 added Boolean operators for the search keys. Users can even create stop lists to exclude common but useless words from indexing. The database also supports query expansion; that is, it retrieves documents containing words of the documents shown by a query. When tested, however, the query speed left a lot to be desired. And as every full-text document ends up in the database, the database can soon become unwieldy.

The *DBIx::FullTextSearch* Perl module, which defines an index of its own using MySQL as its back-end, also has a few


mschilli@mybox:/mnt/big2/mschilli.do.not.delete/DEV/articles/desktop

```
mysql> show columns from file:
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| fileid     | int(11)      |      | PRI | NULL    | auto_increment |
| path       | varchar(255) | YES  | MUL | NULL    |                |
| size       | int(11)      | YES  |     | NULL    |                |
| mtime      | datetime     | YES  |     | NULL    |                |
| atime      | 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)

mysql> 
```

Figure 1: The schema for the 'file' table, in which 'rummage' stores meta-data for files on the filesystem.

quirks. Indexing is a slow process, and it becomes even slower when you have more than 30,000 files in the index.

This is why *rummage* uses the tried-and-trusted SWISH-E indexer, which indexes and searches at an amazing speed. It supports keywords and phrase search and scales really well. The *SWISH::API::Common* module from CPAN facilitates communication with SWISH-E by focusing on the most commonly used aspects. This said, SWISH-E can't delete files from an index once created; and this means reindexing every day to keep up to date. A cronjob running every night can easily handle a couple of hundred thousand files, and that should be quite enough for normal use.

Approaches

After completing an initial indexing session with *rummage -u* (update), users can finally access the meta-data and the full-text index. The command *rummage -k query* finds text files containing a given keyword. Box 1 gives a few examples of different keyword searches and queries for different meta data.

As the schema in Figure 1 shows, the MySQL database stores the full path to every file, its size in bytes, the time and date when it first appeared on the filesystem, the last access time, and the last modification time.

A file named *call.sgml* embedded somewhere in the murky depths of the indexed hierarchy can be found by calling *rummage -p call.sgml*. Under the hood, *rummage* converts *call.sgml* into the SQL pattern *%call.sgml%* and queries the *file* table with *WHERE path LIKE "%call.sgml%"*. Relative paths, such as *examples/call.sgml*, will also work, but in this case, *rummage* will only find the file if it is stored below the *examples* subdirectory.

rummage -n 20 finds the last 20 files that have been modified. If you leave out the integer, the command defaults to the last 10 modified files. *rummage -m "7 day"* gives you all files modified within the last week. To do so, it generates a MySQL query that looks like this

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

telling MySQL to calculate whether the modification date for each entry is more than one week in the past. If needed, you can replace the number of days in the expression with something like *3 month* or *18 hour*. Of course, none of this refers to real time, but to the last database update, which will typically be from the night before. *rummage* just doesn't see anything that happened after this point in time.

You may need to modify the first section in the *rummage* listing to suit your own environment. The *$MAX_SIZE* constant defines the maximum length of the indexed content for a text file. If Perl's -T operator in *SWISH::API::Common* identifies a 100Mbyte logfile as a text file, you will probably not want to index the whole thing. A value of *100_000* specifies that only the first 100Kbytes will be indexed.

One line further down, the DBI-Class module's Data Source Name *$DSN* specifies the database driver (mysql, that is DBD::mysql) and the name of the database (*dts*). Finally, *@DIRS* is an array of directory names, which *rummage* navigates recursively. If symbolic links are used rather than directories, line 24 resolves the links. If indexing your whole home directory takes too long, you can restrict the index to one or multiple subdirectories, such as a local CVS workspace.

Line 27 declares the *psearch* function, which later outputs the search results from the various queries. The function uses a prototype to do this, specifying that *psearch* expects a scalar as its one and only parameter. This is important as the output from the DBI::Class methods *search()* or *search_like()* to *psearch* has to be in a scalar context, as this is the only way to return an iterator that *psearch* can evaluate.

Without the prototype, the *search()* method in the expression *psearch ($db->search(...))* would be in the array context – and this would mean that the *DBI::Class* module's *search()* method

Rummage Commands

```
rummage -u -v                         # Refresh or create database;
                                      # -v for verbose status output
                                      # in the logfile
rummage -k 'linux'                    # keyword search for "linux"
rummage -k '"mike schilli"'           # Search for phrase
rummage -k 'foo AND (bar OR baz)'     # Documents with "foo" and "bar"
                                      # or with "foo" and "baz"
rummage -k 'torvald*'                 # Wildcard search
rummage -p pathlike                   # Search for file by name or path
rummage -n 20                         # Display the last 20 files
                                        modified
rummage -m '7 day'                    # All files modified last week
```

would return a list of matches by definition rather than an iterator.

getopts() analyzes the parameters passed to it. The database update parameter (*-u*) enables the Log4perl framework. If the user specified verbose output (*-v*), the level is set to $DEBUG; the default is $INFO which only stores informational messages in the logfile.

The logfile is overwritten each time to avoid filling up the hard disk. An alternative approach would be to use a Log4perl configuration with *Log::Dispatch::FileRotate*.

In line 41, *db_init()* calls the function with this name in 186; the function initializes the database with the *file* table, if this has not already been done. The function additionally defines an index on the *path* column to allow *rummage* to quickly check later if an entry for a file already exists, and if the timestamp for the file has changed. These extra features mean that the initial *rummage* search after installation can take a while. But don't worry, updates will be a lot quicker later.

Listing 1: rummage

```
001 #!/usr/bin/perl -w
002 ##############################
003 # rummage -  Index and search
004 #           the home directory
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 # Keyword search
060 if ( $opts{k} ) {
061   my @docs = $swish->search(
062                    $opts{k} );
063   print $_->path(), "\n"
064     for @docs;
065
066   # Search by 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   # Search by path
080 } elsif ( $opts{p} ) {
081   psearch(
082     $filedb->search_like(
083       path => "%$opts{p}%"
084     )
085   );
086
087   # Search newest
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
100   psearch(
101     $filedb->search_newest()
102   );
103
104   # Index Home Directory
105 } elsif ( $opts{u} ) {
106   # Uncheck all documents
107   $filedb->set_sql(
108     "uncheck_all", qq{
109     UPDATE __TABLE__
110     SET checked=0
111   });
112   $filedb->sql_uncheck_all()
113     ->execute();
114
115   find( \&wanted, @DIRS );
116
117   # Update keyword index
118   $swish->index_remove();
119   $swish->index(@DIRS);
120
121   # Delete all dead documents
122   # in the DB
123   $filedb->set_sql(
```

Class::DBI::Loader connects to the database in line 44 to generate the object-oriented representation of the database for Class::DBI. Following this, object-oriented access to the *file* table occurs using the *Rummage::File* class. If any of the *search()* calls returns an iterator, it is output via *psearch()*, which simply calls - > *next()* until the iterator does not return any more results. A result object's *path()* method retrieves the file path for each match, while the *mtime()* method retrieves the last modification time for the entry.

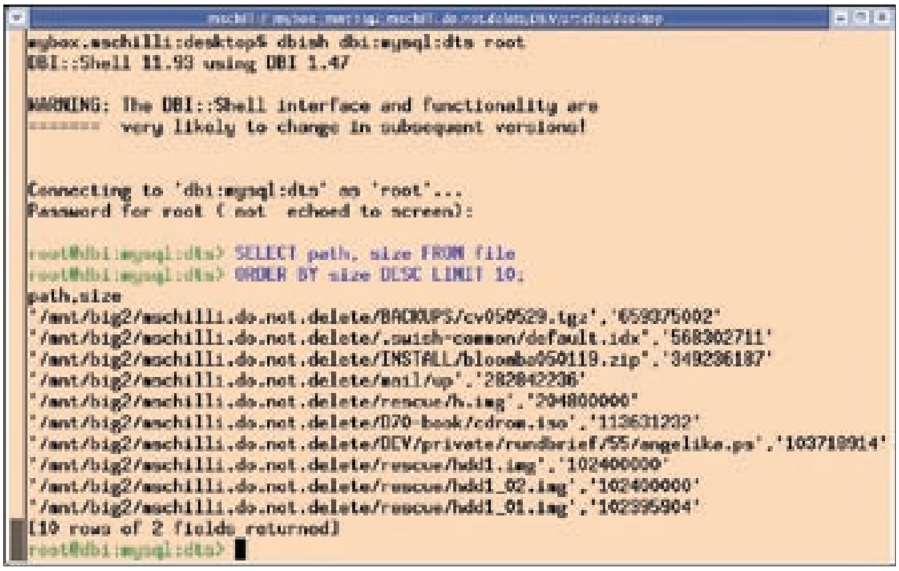
Not all queries can be easily performed using a *Class::DBI* abstraction. When things start to get more complicated, you can drop down to SQL level with Class::DBI. The *set_sql* method allows you to define queries, such as *newest* in line 92, which is then available in the Class::DBI abstraction as *search_newest()*.

Up to Date

When *rummage* sees the *-u* parameter on the command line, it will search the

Listing 1: rummage

```
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 "usage: $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 
160     if ( $entry->mtime() eq
161       $mtime ) {
162       DEBUG "$fn unchanged";
163     } else {
164       INFO "$fn changed";
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 "DB conn failed: ",
196     DBI::errstr unless $dbh;
197 
198   if ( !$dbh->do(
199       q{select * from
200         file limit 1}
201     )) {
202     $dbh->do( q{
203   CREATE TABLE file (
204     fileid      INTEGER
205              PRIMARY KEY
206              AUTO_INCREMENT,
207     path       VARCHAR(255),
208     size       INTEGER,
209     mtime      DATETIME,
210     atime      DATETIME,
211     first_seen DATETIME,
212     type       VARCHAR(255),
213     checked    INTEGER
214   })) or LOGDIE
215       "Cannot create table";
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 }
```



```
mybox.mschilli:desktop$ dbish dbi:mysql:dts root
DBI::Shell 11.93 using DBI 1.47

WARNING: The DBI::Shell interface and functionality are
=======  very likely to change in subsequent versions!


Connecting to 'dbi:mysql:dts' as 'root'...
Password for root ( not  echoed to screen):

root@dbi:mysql:dts> SELECT path, size FROM file
root@dbi:mysql:dts> ORDER BY size DESC LIMIT 10;
path,size
'/mnt/big2/mschilli.do.not.delete/BACKUPS/cv050529.tgz','659375002'
'/mnt/big2/mschilli.do.not.delete/.swish-common/default.idx','568302711'
'/mnt/big2/mschilli.do.not.delete/INSTALL/bloombo050119.zip','349236187'
'/mnt/big2/mschilli.do.not.delete/mail/up','282842236'
'/mnt/big2/mschilli.do.not.delete/rescue/h.img','204800000'
'/mnt/big2/mschilli.do.not.delete/070-book/cdrom.iso','113631232'
'/mnt/big2/mschilli.do.not.delete/DEV/private/rundbrief/55/angelika.ps','103718914'
'/mnt/big2/mschilli.do.not.delete/rescue/hdd1.img','102400000'
'/mnt/big2/mschilli.do.not.delete/rescue/hdd1_02.img','102400000'
'/mnt/big2/mschilli.do.not.delete/rescue/hdd1_01.img','102395904'
[10 rows of 2 fields returned]
root@dbi:mysql:dts> 
```

Figure 2: Using a MySQL query to locate the biggest disk space hogs.

filesystem using File::Find, and add the latest meta-information to the database. To start off, the *UPDATE* command, which is defined in line 107 and run in line 112, sets the *checked* column value for all table entries to 0. If the search function does find an entry in the filesystem, this entry is tagged as verified by setting the *checked* column for the entry to 1. Any entries left with a value of *checked = 0* after completing the search have obviously disappeared from the filesystem since the last search; these entries need to be deleted from the database and removed from the full text index.

Line 115 launches the *find* function, which starts searching the specified directories and digs down through the file structure. The *wanted* function defined in line 140 is called whenever an entry is found. Line 142 immediately drops anything that does not look like a file. The *stat* command in line 150 discovers the file size in bytes, along with the last read and write times associated with the file.

If an entry matching the path is found in the database, line 160 checks if the last modification time is identical to the value for the modification time stored in the database. If the modification times are not identical, lines 165 through 167 update the meta-information (*mtime*, *atime*, *size*) for the entry. If the file is not already in the database, the *create* method in line 170 creates a new entry. The call to *checked()* in line 180 sets the *checked* field to 1, followed by *update()*, which actually performs the update transaction.

Time Format Conversion

MySQL expects "YYYY-MM-DD HH:MM:SS" formatted DATETIME fields, but the Perl *stat* command returns the Unix time in seconds. The *Time::Piece::MySQL* module provides the *mysql_datetime* method to convert the value returned by Perl's time() function to MySQL's time format. The *mysqltime* function defined in *rummage* in line 238 shortens the call.

Garbage and Disk Space Hogs

Users can play around with the metadata for files that *rummage* has processed with the *mysql* client program before adding more intelligence to *rummage* with *DBI::Class*-based queries.

The *dbish* DBI shell from CPAN connects to any database supported by DBI and supports SQL queries. It is installed with the *DBI::Shell* module from CPAN. The following call is for a MySQL database: *dbish dbi:mysql:<TABLE> user password*. Figure 2 shows the shell in action: a SQL query for the ten biggest disk space hogs:

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

will have the culprits squealing for mercy.

The following SQL expression finds the ten oldest files that have not been touched for years:

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

Text files are processed by the indexer every day. Unless you mount the filesystem with the *noatime* option set, the last access date is never more than one day in the past.

Installation

The CPAN shell should guide you through the installation of the required Perl modules. The *mysqladmin* tool will help you create the *dts* database in MySQL: *mysqladmin --user=root create dts*. *rummage* takes care of the database tables automatically. A cronjob calls *rummage* once a day at 3:05 am:

05 03 * * * LD_LIBRARY_PATH=/usr/local/lib /home/mschilli/bin/rummage -u -v >/dev/null 2>&1

The MySQL database is included with most Linux distributions. You can also download it from mysql.com.

The *swish-e* indexer and the *SWISH::API* module are available from swish-e.org. *SWISH::API::Common* from CPAN attempts to install both automatically. If this does not work, you might prefer to download swish-e 2.4.3 or newer, and then run *./configure; make install* to install. The *SWISH::API* module is included with the distribution. The following commands

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

handle the installation. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/59/Perl*

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

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.