Search This!

By Colin Viebrock

Table of Contents

Installing ht://Dig

So, you want to put a "Search This Site" button on your PHP3 sites? Well, unless all your pages are stored in a database (ugh!), there is no easy way to do this using PHP alone.

The solution is to use another program designed specifically for indexing and searching your site, and then use the results of that program in your PHP script.

When I first needed this functionality for a site I was designing for the SummerWorks Theatre Festival, I came across ht://Dig. For those who don't know, "The ht://Dig system is a complete world wide web indexing and searching system for a small domain or intranet. ... [It] is meant to cover the search needs for a single company, campus, or even a particular sub section of a web site."

This tutorial will cover the following steps needed to get these two great Open Source applications working together:

1. Installing ht://Dig
2. Configuring ht://Dig
3. Indexing the Site
4. Building the Search Page
5. Performing the Search
6. Parsing and Displaying the Results

Installing ht://Dig

I'm not going to spend much time on this section, since the ht://Dig documentation can answer any questions much better than I can in this tutorial.

As of this writing, the latest version of ht://Dig is 3.1.0b4. The latest version can always be found at http://www.htdig.org.

Once you've downloaded the archive, and unzipped it, read the *README* and *htdocs/install.html* files for the latest information and instructions on how to install. I'm running Linux 2.0.35 and Apache 1.3.3, so your set−up may vary, but the first step is to run the configuration program.

```
% ./configure
```

You may then want to edit the *CONFIG* file. Specifically I changed these four settings to the following values:

```
prefix =      /usr/local/htdig
CGIBIN_DIR =  /usr/local/htdocs/htdig/cgi-bin
IMAGE_DIR =   /usr/local/htdocs/htdig
SEARCH_DIR =  /usr/local/htdocs/htdig
```

The last two settings aren't too important, because we're going to be making our own search script and

interface using PHP3.

Once the *CONFIG* file is to your liking, run make and then make install:

```
% make % make install
```

If all went well, ht://Dig should be installed and we can move on to configuring it to work with PHP.

Configuring ht://Dig

Before going any further, it would help if you have a clear idea of how ht://Dig works. There is a very nice explanation on the ht://Dig website (follow the link to "How it Works" on the left side), but here's the abridged version.

The ht://Dig system performs three major tasks that should be performed in the following order:

Digging

Before you can search, a database of all the documents that need to be searched has to be created.

Merging

Once the document database has been created, it has to be converted to something that can be searched quickly. Also, if you want to only update changed documents, these changes have to be merged into the searchable database.

Even though this task could be performed at the same time as digging, it is a separate process for efficiency reasons. It also gives more flexibility to what actually happens at merge time.

Searching

Finally, the databases that were created in the previous steps can be used to perform actual searches. Normally, searches will be invoked by a CGI program which gets its input from the user through an HTML form ... but we'll be doing it all with PHP3!

So, let's see what the installation created, shall we?

% cd /usr/local/htdig/ % ls bin common conf db

The `bin/` directory contains the executables required by ht://Dig. Configurations files are in `conf/` (believe it or not!) and I think you can guess where the database files are.

[Note: I'm going to assume from now on the you installed ht://Dig in the `/usr/local/htdig` directory as shown above, and that the path to your website files is `/www`, probably a symlink to `/usr/local/htdocs`.]

If your web server is like mine, you're probably running several web sites off the same machine. In this case, I've found it useful to have separate configuration files for each site. ht://Dig installed a sample configuration file called `htdig.conf`, so I recommend you copy it to a new filename and make some changes. Here is the `sw98.conf` file I used for the SummerWorks Theatre Festival's website:

---

database_dir: /usr/local/htdig/db/sw98 start_url: http://www.summerworks.on.ca/ limit_urls_to: http://www.summerworks.on.ca/ exclude_urls: /staff/ /search/ .inc .doc .mcw

```
search_algorithm: exact:1.0 endings:0.3 matches_per_page: 50 excerpt_length: 200
template_map: sw98 sw98 /www/summerworks/search/results−template.html
search_results_header: /www/summerworks/search/results−header.html
search_results_footer: nothing_found_file: /www/summerworks/search/results−nomatch.html
syntax_error_file: /www/summerworks/search/results−syntaxerror.html star_image:
http://www.summerworks.on.ca/gifs/x−star.gif star_blank:
http://www.summerworks.on.ca/gifs/x−nostar.gif max_stars: 5 max_doc_size: 100000
valid_punctuation: .−_/!#$%^&*'«»"
```

What do all these settings mean? Some are obvious, but the following probably are not:

database_dir:
Normally all database files are in the `/usr/local/htdig/db/` directory. To keep things a bit cleaner, I've made sub−directories for all the sites on my server, so this is the sub−directory for the SummerWorks site. You'll need to make this directory manually before you can index the site:

```
% cd /usr/local/htdig/db % mkdir sw98
```

exclude_urls:
There are some files we don't want to search. If a URL contains any of the space separated patterns, it will not be indexed. In this case, I didn't want the staff pages, nor the search pages indexed. I also wanted it to ignore `.inc` files (common PHP code that I include on various pages), and some Word documents for PC and Mac. On the real SummerWorks site, there are a couple more exclusions, but you get the idea (I hope!).

search_algorithm:
ht://Dig lets you turn "fuzzy" searching on or off. When it's on, people who search for "play", for example, will also find things like "plays", "player", "players", etc.. This slows down the digging process a fair amount, but can be useful. The different weights applied to each algorithm mean, in this case, that a search for "play" will turn up "player", but not as high in the results as an exact match for "play".

template_map:
This makes a new template called "sw98". More on templates below.

search_results_header, **search_results_footer**, **nothing_found_file**, and **syntax_error_file**:
These are files used by the template, and parsed by PHP. More below.

max_doc_size:
This is bigger than usual, since there are some big pages on the site. If you find that some of your pages aren't being indexed, setting this to a higher value will often solve the problem.

valid_punctuation:
I basically only added the French quotes and the single quote to the list of valid punctuation. This is the set of characters which will be deleted from the document before determining what a word is. This means that if a document contains something like "Andrew's" the digger will see this as "Andrews".

The same transformation is performed on the keywords sent to the search engine.

There are plenty of other setting you can set in the `.conf` file. For a definitive list with explanations, check out the ht://Dig website.

Templates

ht://Dig makes extensive use of templates for it's output. These templates are (usually) plain HTML documents containing variables, which are substituted with the search results.

There are four "standard" template setting in the `.conf` file:

search_results_header:
This specifies a filename to be output at the start of search results.

search_results_footer:
This specifies a filename to be output at the end of search results.

nothing_found_file:
This specifies the file which contains the text to display when no matches were found.

syntax_error_file:
This points to the file which will be displayed if a boolean expression syntax error was found.

There is also the result template file. This is the file referenced in the **template_map** attribute of the `.conf` file. This is where all the information you want the search to return is displayed.

I said that these templates usually contain a bunch of HTML, with some variable substitution. When using ht://Dig with PHP, it would be much easier if a search returned the raw results of the search. You can then use PHP to parse and display that information however you want.

The easiest way to do this is to not put any HTML in the template files, but instead just put in the list of variables you want substituted.

So, make your `results-header.html` file contain only the following 4 lines:

```
$(MATCHES) $(FIRSTDISPLAYED) $(LASTDISPLAYED) $(LOGICAL_WORDS)
```

`results-nomatch.html` should contain only 1 line (note that it's not a variable):

```
NOMATCH
```

`results-template.html` consists of 4 lines:

```
$(TITLE) $(URL) $(PERCENT) $(EXCERPT)
```

Finally, `results-syntaxerror.html` consists of 2 lines:

```
SYNTAXERROR $(SYNTAXERROR)
```

It's important that there be no extra line breaks at the beginning or end of your files, since we're going to rely on the fact that every fourth line of the `results-template.html` file (for instance) is the title of the document.

All of the template files are in a sub-directory of the website called `search/`.

If you want to find out what all the variables mean (and what other ones are available), check the ht://Dig site.

Indexing the Site

Before ht://Dig can search your site, it has to index it.

ht://Dig retrieves HTML documents using the HTTP protocol and gathers information from these documents which can later be used to search them. In this way, it works very much like a search robot or web spider.

[Note: ht://Dig can also operate locally, indexing a site through the local file system. While this is a faster way of indexing a site, it doesn't work very well when indexing dynamic pages. Why? Well, it would indexing the **source** of a PHP file, not the result. And that's not what you want.]

Every time you make a change to your site, you'll want to re−index it. So it's probably a good idea to write a little shell script that indexes your site for you, and add it to your crontab. Here is `rundig.sh`, a script that does just that for the SummerWorks site, and emails me the details. The changes you need to make for your site should be obvious.

#! /bin/sh if ["$1" = "−v"]; then verbose="−v" fi # This is the directory where htdig lives BASEDIR=/usr/local/htdig # This is the db dir DBDIR=$BASEDIR/db/sw98 # This is the directory htdig will use for temporary sort files TMPDIR=/tmp export TMPDIR # This is the name of a temporary report file REPORT=$TMPDIR/htdig.sw98 # This is who gets the report REPORT_DEST="you@your−email−address.com" export REPORT_DEST # This is the subject line of the report SUBJECT="ht://Dig Report for SW98" # This is the name of the conf file to use CONF=sw98.conf # This is the PATH used by this script. Change it if you have problems # with not finding wc or grep. PATH=/usr/local/bin:/usr/bin:/bin ##### Dig phase STARTTIME=`date` echo Start time: $STARTTIME echo rundig: Start time: $STARTTIME > $REPORT $BASEDIR/bin/htdig $verbose −s −a −c $BASEDIR/conf/$CONF >> $REPORT TIME=`date` echo Done Digging: $TIME echo rundig: Done Digging: $TIME >> $REPORT ##### Merge Phase $BASEDIR/bin/htmerge $verbose −s −a −c $BASEDIR/conf/$CONF >> $REPORT TIME=`date` echo Done Merging: $TIME echo rundig: Done Merging: $TIME >> $REPORT ##### Cleanup Phase # To enable htnotify, uncomment the following line # $BASEDIR/bin/htnotify $verbose >>$REPORT # To enable the soundex or endings search, uncomment the following line $BASEDIR/bin/htfuzzy $verbose −c $BASEDIR/conf/$CONF endings # Move the work files mv $DBDIR/db.wordlist.work $DBDIR/db.wordlist mv $DBDIR/db.docdb.work $DBDIR/db.docdb mv $DBDIR/db.docs.index.work $DBDIR/db.docs.index mv $DBDIR/db.words.db.work $DBDIR/db.words.db END=`date` echo End time: $END echo rundig: End time: $END >> $REPORT echo # Grab the important statistics from the report file # All lines begin with htdig: or htmerge: or rundig: fgrep "htdig:" $REPORT echo fgrep "htmerge:" $REPORT echo fgrep "rundig:" $REPORT echo WC=`wc −l $REPORT` echo Total lines in $REPORT: $WC # Send out the report ... mail −s "$SUBJECT − $STARTTIME" $REPORT_DEST < $REPORT # ... and clean up rm $REPORT

Run this from the command line with the −v switch and you can watch as your site is indexed! You'll need to run this as root (or the same user you installed ht://Dig as) so that it can create the necessary files in `/usr/local/htdig/db`.

Building the Search Page

Now that your site is being indexed regularly (you did put an entry for `rundig.sh` in your crontab, right?), you can search the site.

Your search page is just going to be a simple HTML form:

```
<FORM METHOD="POST" ACTION="results.php3"> Search for: <INPUT TYPE="text"
NAME="search" SIZE=30> <INPUT TYPE="submit"> </FORM>
```

Performing the Search

ht://Dig is normally run as a CGI script ("normally" meaning "by everyone who doesn't use PHP3!"). To run the search program from PHP, you will need a file called `htdig.sh` which looks like this:

```
#!/bin/sh HTBINDIR=/usr/local/htdocs/htdig/cgi−bin QUERY_STRING="$@"
REQUEST_METHOD=GET export QUERY_STRING REQUEST_METHOD
$HTBINDIR/htsearch
```

Just change the value of `HTBINDIR` to whatever is appropriate for your setup (i.e. the path you picked for `CGIBIN_DIR` in the `CONFIG` file when you installed ht://Dig).

I recommend putting this file in the `search/` subdirectory as well, along with the template files. You also need to change file permissions so `htdig.sh` is executable by your web server.

Next, we need to get PHP to talk to the search script. Finally, some PHP code! This is simply a matter of building the query string to pass to `htdig.sh`, executing the search, and parsing the results.

So, in `results.php3`, we build the query string:

```
<?php $HTSEARCH_PROG = "/www/summerworks/search/htdig.sh"; $words =
EscapeShellCmd(UrlEncode($search)); $config = "sw98"; $format = "sw98"; $query =
"config=$config&format=$format&words=$words"; ?>
```

$HTSEARCH_PROG is the location of the `htdig.sh` program. *$search* is passed from the form, and is the list of words to search for. *$config* tells ht://Dig which `.conf` file to use, and *$format* tells it which template map to use from the `.conf` file.

Then we run the search:

```
<? $command="$HTSEARCH_PROG \"$query\""; exec($command,$result); ?>
```

This puts the results of the search into the array *$result*, which PHP then parses to display the results.

Displaying the Results

This is where ht://Dig's templates are put to work. First, let's count the number of lines returned by the search:

```
<?php $rc = count($result); ?>
```

The first two lines are always going to be the HTTP header

```
Content−type: text/html \n \n
```

so we can skip them (remember, ht://Dig was designed to output HTML).

If the search produced an error, we'd get no results or a one−line error, so:

```
<?php if ($rc<3): echo "There was an error executing this query. Please try later.\n"; ?>
```

Let's also check to see that we got some matches. If we didn't, then ht://Dig will just echo out the two−line HTTP header, then the contents of the `results−nomatch.html` file. We use that information to perform the check:

```
<?php elseif ($result[2]=="NOMATCH"): echo "There were no matches for <B>$search</B>
found on the website.<P>\n"; ?>
```

We know to look for "NOMATCH" because that's the string in results−nomatch.html).

Similarly, we can check for a boolean syntax error:

```
<?php elseif ($result[2]=="SYNTAXERROR"): echo "There is a syntax error in your search
for <B>$search</B>:<BR>"; echo "<PRE>" . $result[3] . "</PRE>\n"; ?>
```

If none of the above conditions were true, then we have at least one match! In this case, ht://Dig first outputs the variables in the results−header.html template, so:

```
<?php else: $matches = $result[2]; $firstdisplayed = $result[3]; $lastdisplayed = $result[4];
$words = $result[5]; echo "Your search for <B>$words</B> returned <B>$matches</B>
match"; echo ($matches==1) ? "" : "es"; ?>
```

I'm echoing out the variable *words* here, instead of *$search*. That's because if I have fuzzy searching turned on and I search for "play", then *$search* will equal "play", but *$words* will equal "(play or played or playing or player or plays or players)" ... which what you are really searching for.

Now we echo out each of the matches.

```
<?php $i=6; ?>
```

(6 is the number of variables in results−header plus 2)

```
<?php while($i<$rc) { # grab the match information $title = $result[$i]; $url = $result[$i+1]; $percent = $result[$i+2]; $excerpt = $result[$i+3]; # output the match information echo "<A HREF=\"" . $url . "\">" . $title . "</A><BR>\n"; echo "(" . $percent . "% match)<BR>\n"; echo "<blockquote>" . $excerpt . "</blockquote><BR><BR>\n"; # move to the next match $i = $i + 4; } endif; ?>
```

And that's it!

There are a few quirks to keep in mind. Notably, ht://Dig outputs an additional new−line after the $(STARSLEFT) and $(STARSRIGHT) variables in the template. You need to keep this in mind when figuring out which line of *$result* corresponds to what piece of information. For example, if I changed $(PERCENT) to $(STARSLEFT) in my `.conf` file, I would need to make the following changes to the code:

```
<?php $title = $result[$i]; $url = $result[$i+1]; $stars = $result[$i+2]; $excerpt = $result[$i+4]; ... $i=$i+5; ?>
```

But the basic strategy is always the same: PHP loops through the array *$result* and outputs the information. If you want to see the contents of *$result* for yourself (to make sure you're getting the right results), just replace the entire last else− block above with:

```
<?php while (list($k,$v)=each($result)) { echo "$k −> $v \n"; } ?>
```

Conclusion and Advanced Topics

You should all be running out now and adding "Search this Site" buttons to your PHP−driven web pages. This tutorial covered the basics, but there are a lot of advanced things you can do with ht://Dig and PHP:

- show matches in groups (e.g. "Matches 1−10", "Matches 11−20")
- "fix" the URL for matches for sites that are in framesets
- index and search not only HTML and PHP pages, but also PDF (Acrobat) and Microsoft Word documents using external parsers

- use the raw results of the indexing process to generate a site−map

Look for more tutorials later on these and other topics.

But until then, happy searching!