Apropos Replacement : Development of a full text search tool for man pages

Abhinav Upadhyay
<*er.abhinav.upadhyay@gmail.com*>

Jörg Sonnenberger
<*joerg@NetBSD.org*>

Abstract

Manual pages are a key component of Unix like operating systems, they have played a significant role in the success of Unix over the years. Man pages are the most authentic source of reference for a system administrator or a programmer. However, there has been a lack of good search tool to make man pages more accessible.

Traditionally, *apropos(1)* has been there as a search interface but it was developed in the early days of Unix when computing resources were scarce and that is the primary reason for its simple design and limited search capabilities.

This paper discusses a new implementation of *apropos* done by Abhinav Upadhyay and mentored by Jörg Sonnenberger as part of Google Summer of Code 2011, for The NetBSD Foundation. The goal of this project was to replace the conventional apropos in NetBSD with a modern version which supports full text searches.

1 Introduction

The classical version of *apropos* has been implemented by simply indexing the keywords in the NAME section of the man pages in a plain text file (whatis.db) [2] and performing searches on it. The reason for this simple design was most probably lack of computing resources in the early days. The plain text file consisting of keywords hardly takes few hundreds of kilo bytes of disk space and performing search on a plain text file is quite easy.

This simplified design of *apropos* resulted in limited utility and usability. Only if the users know the exact keywords do they get the appropriate results; otherwise most of the times the searches are full of irrelevant results or possibly a dead end. In the modern computing world, where hard problems like searching the World Wide Web have been solved [3] to a sufficient degree, it makes perfect sense to leverage the advancement in technology in order to come up with a better search tool for man pages.

In present times, the machines are powerful and capable of running the search algorithms efficiently. Disk space is also sufficiently cheap that any extra space incurred by indexing of additional metadata from man pages can be easily afforded. These are necessary prerequisites for building an effective search tool.

In this paper, the limitations of the traditional implementation of apropos are discussed along with brief details of how these were overcome as part of this project. Additionally, a comparison of other modern implementations of apropos has been done.

2 Limitations of Conventional *apropos*

2.1 Lack of support for free form queries

As noted earlier the conventional *apropos* is limited to the keywords used in the NAME section of the man pages. This means that the users do not have a whole lot of options for specifying keywords in their queries. Besides this, they cannot search for keywords which are not usually specified in the title of a man page, for example a query like `EINVAL` would return no results. See **listing 1** for concrete examples.

2.2 Lack of basic language support

Another major limitation of the classical version of *apropos* is that it is not smart enough to provide support for basic natural language processing constructs, like stemming or spelling corrections. As it can be seen in **listing 2**, while *apropos* returns the correct result for the query *"make directories"*, it fails when the keyword *"directory"* is used in place of *"directories"*, even though the two words are based on same root word.

Similarly, it is very common for users to misspell keywords in a query and *apropos* has no support for detecting it and offering any useful behavior either (see **listing 3**).

2.3 Unintelligible Output

Because of the way *apropos* works, sometimes its output can be unintelligible and it can be very hard for the user to identify relevant results, see **listing 4** for an example.

2.4 Other Problems

Apart from the search related problems there are a few issues related to the way man pages are handled in NetBSD. The different aliases of the man pages are stored on the file system in the from of hard (or symbolic) links and these have to be specified in the makefiles explicitly using the `MLINKS` mechanism. This approach works fine but it is a mess from maintenance point of view. It should be possible to fix this by utilizing the index already built and maintained by *apropos*, but yet again, the simplistic implementation of *apropos* does not leave any room for improvement.

3 Solution: A new apropos

This project proposes a very simple and straightforward solution to solve these problems and make man pages more usable. The idea is to exploit the inherent semantics and structure of the man pages. The *mdoc(7)* [21] and *man(7)* [16] macros used for writing man pages are semantically quite rich, those macros combined with the context of the surrounding section of the man page in which they have been used, can act as an indicator of the relevancy of the content. For example, the `.Nm` macro when used inside of the `NAME` section holds more significance than if it is used elsewhere. Similarly the description of various *errno* values in the *intro(2)* man page has more important information than the mention of those *errno* values in the `RETURN VALUES` section of various man pages from section 3.

In essence, it should be possible to parse and extract the structurally meaningful data out of man pages and index them in the form of an inverted index [4]. The index can be combined with a ranking algorithm which exploits the structured nature of the data to give more relevant results to the user, and to build a high quality search interface for manual pages. Such a search tool will solve many of the problems associated with the conventional *apropos(1)* as noted before and discussed as follows:

Free Form Queries

The users can express their queries in more natural language form as the searches are no longer limited to the the NAME section. For example, queries like *"installing new software package"* may now produce relevant results.

Basic Natural Language Processing Support

When parsing the man pages and building the index, tokens are preprocessed after extraction from the man pages to support some of the very basic natural language processing functionalities. In this implementation, the Porter stemming algorithm [5] has been used to reduce the individual tokens extracted from the man pages to their root words. This enables support for more flexible searches. For example both *"Installing new packages"* and *"install new package"* will return same results.

Similarly along with the inverted index, a dictionary of the keywords frequently occurring in the corpus of man pages is also built and used to support spelling suggestion.

Bookkeeping of man page metadata

In this implementation, additional metadata related to the man pages, for example an MD5 hash, the device id, inode number and modification time of the man page files are kept in the database. This allows fast incremental update of the index as new man pages are installed or the old ones are updated.

Similarly a separate index of all the man page aliases is stored and maintained. This provides an option to get rid of all the hard or symbolic links of the man pages scattered throughout the filesystem, and also for clean up of the `MLINKS` mess in the makefiles.

3.1 Tools Used

The two main operations that are critical for this project are parsing of the man pages and building of an index of the data obtained from the parsing process. These are highly specific problems, and solving them from scratch

```
$ apropos “add new user”
add new user: nothing appropriate

$ apropos “get process status”
get process status: nothing appropriate

$ apropos “termcap database”
termcap database: nothing appropriate
```

Listing 1: No results for free form queries

```
$ apropos “make directories”
mkdir (1) - make directories

$ apropos ”make directory”
make directory: nothing appropriate

$ apropos “upgrading package”
pkg_add (1) - a utility for installing and upgrading software package distributions

$ apropos “upgrading packages”
upgrading packages: nothing appropriate
```

Listing 2: No support for stems or words with same root

```
$ apropos “copy strings”
stpcpy, stpncpy, strcpy, strncpy (3) - copy strings

$ apropos “coppy strings”
coppy strings: nothing appropriate
```

Listing 3: No spelling correction

```
$ apropos power
PCI, pci_activate, pci_bus_devorder, pci_chipset_tag_create,
pci_chipset_tag_destroy, pci_conf_read, pci_conf_write,
pci_conf_print, pci_conf_capture, pci_conf_restore, pci_find_device,
pci_get_capability, pci_mapreg_type, pci_mapreg_map, pci_mapreg_info,
pci_intr_map, pci_intr_string, pci_intr_evcnt, pci_intr_establish,
pci_intr_disestablish, pci_get_powerstate, pci_set_powerstate,
pci_vpd_read, pci_vpd_write, pci_make_tag, pci_decompose_tag,
pci_findvendor, pci_devinfo, PCI_VENDOR, PCI_PRODUCT, PCI_REVISION (9)
- Peripheral Component Interconnect
PMF, pmf_device_register, pmf_device_register1, pmf_device_deregister,
pmf_device_suspend, pmf_device_resume, pmf_device_recursive_suspend,
pmf_device_recursive_resume, pmf_device_resume_subtree,
pmf_class_network_register, pmf_class_input_register,
pmf_event_inject, pmf_set_platform, pmf_get_platform (9) - power
management and inter-driver messaging framework
acpi (4) - Advanced Configuration and Power Interface
acpipmtr (4) - ACPI Power Meter
amdpm (4) - AMD768 Power Management Controller and AMD8111 System
Management Controller
...
.
.
.
```

Listing 4: Unintelligible output of *apropos(1)*

would be a separate project in its own right, therefore this project tried to avoid reinventing the wheel and used existing tools and libraries which are battle tested and very successful. *libmandoc* [6] from the *mdocml* [18] project has been used for parsing the man pages and *sqlite* [7] for indexing and storing the parsed data on the file system.

libmandoc

libmandoc is a library interface to a validating compiler for man pages. It provides interface to parse and build an AST (Abstract Syntax Tree) of the man page. It also provides an interface for traversing that tree in order to extract the data from its nodes.

sqlite

sqlite is an embedded relational database management system providing a relatively small and easy to use C library interface. One of the main reasons for choosing it over the myriad of other possible options is that it provides built in support for full text search through its FTS virtual table module [8]. The FTS module can be accessed using pretty much standard SQL syntax, and it is still flexible enough to accept user supplied ranking function to suit the needs of the application. Besides that, another advantage of *sqlite* is that, being an RDBMS, it makes it very easy to store additional metadata in the form of normal database tables without any hassles.

4 Implementation Details

4.1 makemandb

makemandb(8) [9] is the key component of this implementation. It is a command line tool which traverses the filesystem, reads the raw man page source files, extracts the interesting data from them using the *libmandoc* parser and then stores that data in an FTS table using *sqlite*.

Operation of makemandb

makemandb uses a two pass algorithm to index the man pages. In the first pass it obtains the list of directories containing man page files from *man.conf* and starts traversing those directories. It obtains the $\{dev\_t,\ ino\_t,\ mtime\}$ of all the files using *stat(2)*. A similar set of metadata is stored in the database (referring to the last successful indexing operation), *makemandb* takes a difference (the set difference operation) of the two sets of metadata to obtain the list of man page files on the file system which are either newly installed (new $\{dev\_t,\ ino\_t\}$ pairs) or they have been updated (changed *mtime*).

In the second pass *makemandb* generates MD5 checksum of the files obtained from the previous stage and for each of these checksums, it checks in the db, whether this MD5 checksum already exists in the db or not. If the MD5 checksum is already present in the database, then this was just a false alarm and *makemandb* does not need to parse the file, it simply updates the metadata of that file in the database. In case the MD5 checksum of a file does not already exist in the database, then that means the file is either a newly installed man page or an older copy of a man page was updated, *makemandb* feeds such man pages to the parser and extracts interesting data out of them, later on storing the data in the FTS table in the database.

Database Schema

During the initial stage of the project, the structure of the database was kept simple. The FTS table consisted of 3 columns: `name`, `name_desc`, and `desc`. The `name` and `name_desc` columns stored the name of the man page and the one line description from the `NAME` section respectively, while the `desc` column stored the content from the rest of the sections. This approach worked quite well for a small set of man pages (for example the 4000 or so man pages in the base set of NetBSD). However, as the number of documents were increased (for example including man pages from pkgsrc), the quality of search started deteriorating. This was mainly because bulk of the parsed data was being stored in the same column in the database and the ranking algorithm had no way of identifying more relevant results except from identifying whether the match was found in the `NAME` section or any other section.

To rectify this, first there was an attempt to try out better ranking schemes (the various ranking schemes that were tried out during the development of the project are discussed later). Better ranking schemes required storing precomputed term weights [19] on the file system, this almost doubled the storage requirements, therefore such ranking schemes were discarded later on, and the database schema was decomposed further into more columns. The decomposition into more columns was done to represent the most commonly occurring sections in usual Unix man pages, for example `NAME`, `DESCRIPTION`, `LIBRARY`, `RETURN VALUES`, `EXIT STATUS`, `ERRORS` etc. Such a decomposition allowed for an elaborate ranking

```
The Database has three tables:
1. mandb
2. mandb_meta
3. mandb_links

(1) mandb:
    The main FTS table which contains all the content parsed from man pages

  COLUMN NAME       DESCRIPTION
  1. section        Section number
  2. name           The name of the man page
  3. name_desc      Short description from NAME section
  4. desc           Contains the content from DESCRIPTION section and any other
                    section which is not stored in a separate column.
  5. lib            The LIBRARY section
  6. return_vals    RETURN VALUES section
  7. env            ENVIRONMENT section
  8. files          FILES section
  9. exit_status    EXIT STATUS section
  10. diagnostics   DIAGNOSTICS section
  11. errors        ERRORS section
  12. md5_hash      An MD5 hash of the contents of the man page (UNIQUE).


(2) mandb_meta:
    This table maintains essential metadata of all the indexed man page files.

  COLUMN NAME       DESCRIPTION
  1. device         (dev_t)Logical device number from stat(2)
  2. inode          (ino_t)Inode number from stat(2)
  3. mtime          Last modification time from stat(2)
  4. file           Absolute path name (UNIQUE constraint)
  5. md5_hash       MD5 Hash of the man page file (UNIQUE)
  6. id             A unique integer ID referring to a page in mandb (PRIMARY KEY)

(3) mandb_links:
    This is an index of all the hard/soft links of the man pages. The list of
    the aliases is usually obtained from the multiple .Nm entries in the NAME
    section of a man page.

  COLUMN NAME       DESCRIPTION
  1. link           The name of the hard/soft link (UNIQUE index)
  2. target         Name of the target page to which the link points
  3. section        The section number
  4. machine        The machine architecture (if any) for which the page is
                    relevant.
```

Listing 5: Database Schema

scheme, which could assign different weights to each of these columns, based on their relevancy. The final schema of the database is shown in **listing 5**

4.2 apropos

apropos was written from scratch to use the FTS index created by *makemandb*. Since, unlike the traditional *apropos*, this version of *apropos* is not limited to searching within only the `NAME` section, the resulting number of search results is usually quite high, therefore new *apropos* employs a sophisticated ranking algorithm to filter out the most relevant results and rank them up. To avoid cluttering the output and make the user interface clean, the new *apropos* displays only the top 10 results by default with options to display more results if required. During the development and testing of the project it has been observed that, for a query, most often than not the top 10 results are sufficient.

Another key aspect of any search application is to filter out or avoid stop-words . Words like *"the", "a", "an", "this"* etc. come under the category of stop-words, these are the words which are *very* commonly used in the language. Stop-words usually skew the search results because of their sheer frequency in the corpus. The usual practice in the information retrieval world is to filter such stop-words while building the index, however, in case of this project, it proved to be non-trivial within the given time constraints. Therefore it was decided that, rather than eliminating stop-words from the index, a reasonable trade off can be made by filtering out any stop-words from the user's search query and querying the database for only the keywords which do not come under the category of stop-words.

4.3 apropos-utils

apropos-utils [10] is a small library interface provided with this implementation. It provides functions for querying the FTS index and for processing the results in a user supplied callback function. Its main purpose is to develop different interfaces on top of it for different use cases. For instance a CGI front end has been built using it for doing the searches from a web browser, similarly an IRC bot was also developed utilizing this interface.

4.4 Ranking Algorithm

The ranking algorithm is the most interesting and most crucial component of any search application, it is the deciding factor of the usefulness of the application. This project took several stabs at coming up with a suitable ranking algorithm for *apropos*.

1.

Initially the database schema was quite simple, it consisted of only three columns `name, name_desc, desc` (as described previously). At this point, there weren't too many documents in the corpus, therefore nothing sophisticated was required. The basic intuition was that, a match found in the `NAME` section of a man page has more weight as compared to a match found in any other section. This ranking scheme worked well for a small set of man pages in the corpus.

2.

As more and more man pages were being added to the corpus for testing purposes, the quality of search results started to dilute. At this point of time, a completely new approach was adopted for ranking the results. In the information retrieval literature, *tf-idf* [19] weights based ranking models are very common. The *tf* in *tf-idf* stands for *Term Frequency* and *idf* stands for *Inverse Document Frequency*.

Term Frequency: The term frequency of a term `t` in a document `d` refers to the count of the number of times the term `t` appears in the document `d`

Inverse Document Frequency: The *idf* of a term `t` is the number of documents in which the term appears *at least once*.

Term frequency is a *local* factor, it is concerned only with the number of occurrences of the search terms in one particular document at a time. While inverse document frequency is a *global* factor, in the sense that it indicates the discriminating power of a term. If a term appears in only a selected set of documents, then that term separates those set of documents from the rest. Therefore, ranking obtained by combining these two factors brings up more relevant documents.

The weight of a term `t` in document `d` is calculated by the following formula:

$$\boxed{weight = tf \times idf}$$

Where *tf* = Term frequency of term `t` in document `d`

Inverse document frequency is calculated using the following formula:

$$\boxed{idf = log(\frac{N}{N_t})}$$

Where N = Total number of documents in the corpus
N_t = Number of documents in which term t occurs (at least once).

So for a term which appears in only one document, it will have $idf = log(n)$, while a term which appears in all the documents, it will have $idf = log(1) = 0$. For example a term like *"the"* will have a high *term frequency* in almost all the documents, but at the same time it will have its *inverse document frequency* almost close to 0, which will nullify its effect on the quality of search results.

A lot of research has been done on *tf-idf* weights based ranking algorithms and various techniques have been proposed over the years. According to a survey done by *Salton* and *Buckley* in 1988 [20], the following ranking scheme was found to be most effective:

$$\boxed{\frac{tf \cdot log(\frac{N}{N_t})}{\sqrt{\sum(tf \cdot log(\frac{N}{N_t}))^2}}}$$

The implementation of the above scheme proved to be *very* slow. It required too many computations and these were required to be done for each search result separately to compute its relevance weight. In a state of the art search application, the index would usually store precomputed *tf-idf* weights on the file system, in which case the overhead of computing weights of each individual search result would require very less work, leading to better performance. An attempt was made to compute the *tf-idf* weights while indexing the man pages and store them in a separate table. This lead to high quality search results but it also brought up a serious issue. Storing precomputed weights on the file system, more than doubled the storage requirements of the database index. As a concrete example, it took close to 90 MB to index about 8000 documents. Without the precomputed weights, the database size requirement was close to 45 MB for the same number of pages.

3.

Since the conventional *apropos* required only a few hundred kilobytes of disk space, the disk space requirements of new *apropos* seemed hyperbolic, therefore it was decided later on to look for alternative ranking schemes. At this point the database schema was decomposed into several columns representing some of the most common sections found in man pages (see **Listing 5**) and a fast yet sophisticated ranking algorithm was adopted, to exploit this structured nature of the data.

Okapi BM25F [15] is considered one of the most successful probabilistic models for information retrieval. It is based on *tf-idf* scheme but it takes into account for the structure of the data. It allows for separate weights to be assigned for different parts of a structured document to come up with better relevance rankings. The exact implementation of this algorithm did not really provide any great results and therefore it had to be adapted and modified to suit the needs of *apropos*, considering the structure of the database schema. The actual algorithm implemented is described in pseudo-code below. *apropos* executes this function for each of the document obtained as part of the result set for the query and computes their relevancy weight. The documents are ranked in decreasing order of their weights.

```
nhitcount =  Number of occurrences of
             phrase p in column c in
             current document

nglobalhitcount = Number of occurrences of
                  phrase p in column c in
                  all the documents.

ndocshitcount = Number of documents in
                which phrase p occurs
                in column c, at least
                once.
```

Listing 6: Definition of important entities used in algorithm 1

Algorithm 1 Compute Relevance Weight of a Document for a Given User Query

```
Require: User query q
Require: Document d whose weight needs to be computed
Require: An array weights consisting of preassigned weights for different columns of the FTS table
tf ← 0.0
idf ← 0.0
k ← 3.5                ▷ k is an experimentally determined parameter
doclen ← length of the current document
ndoc ← Total number of documents in the corpus
for each phrase p in q do
    for each column c in the FTS table do
        $w \leftarrow weights[c]$          ▷ weight for column c
        $idf \leftarrow idf + log(\frac{ndoc}{ndocshitcount}) \times w$
        $tf \leftarrow tf + \frac{(nhitcount \times w)}{(nglobalhitcount \times doclen)}$
    end for
end for
$score \leftarrow \frac{(tf \times idf)}{(k + tf)}$
return score
```

5 Results

5.1 Performance of apropos

Listing 7, **8**, and **9** show results of some of the sample queries on new *apropos* to demonstrate how it performs as compared to the traditional version. [1] In **Figure 1**, the spell corrector can be seen in action along with the web interface that was developed using *apropos-utils*.

5.2 Performance of makemandb

On a modern desktop hardware, *makemandb* takes about 25 seconds to index 4000 man pages, and the size of this index is 15 MB. When the number of man pages is increased to 6100 by including some packages from *pkgsrc*, the time taken by *makemandb* grows to about 50 seconds and the size of the database index grows to 27 MB.

6 Related Work

man-db

man-db [11] is a complete implementation of the man page documentation system and it is used on a number of GNU/Linux distributions. *man-db* takes an interesting approach for indexing the man page data. Unlike the classical *apropos*, it uses a Berkley DB database but still its index is limited to the NAME section only. It adds an option ‘K’ to *man(1)* to allow a crude full text search but it is not very efficient nor effective.

mandocdb

mandocdb [12] is part of the *mdocml* project and it is more in line with the goals of this project but it takes a novel approach. It indexes the keywords extracted from the man pages in a key-value store using btree(3) [13]. It also comes with its own implementation of *apropos* which performs search using this key-value store. The key point of this implementation is that it exploits the semantic structure but rather than indexing the complete content, it plucks out specific pieces of information from the man pages, for example author names, function names, header file names, etc. It is more sophisticated and much more useful than classic *apropos*, but it is still essentially a keyword lookup based implementation.

7 Future Work

Work on Ranking Algorithm

The ranking algorithm is based on the probabilistic model of information retrieval [14]. It uses certain parameters whose values are usually dependent on the corpus and the search application. For example, certain weight parameters have been assigned to different sections of man pages, signifying how important the information in a particular section is as compared to the rest of the sections. At the moment these values have been determined manually but one goal is to use some supervised machine learning techniques in order to automate this to improve the quality of search results further.

Using external storage

A new feature is under development in *Sqlite*, to support external storage option in FTS databases. This option would allow building the FTS index without actually storing any content in the database itself (except for the data required to maintain the indexes). This will save bulk of the disk space currently required by *apropos* database. The disk space thus saved might also allow for experiments with ranking algorithms which require storing additional metadata on the disk.

```
$ apropos “add new user”
ssh-add(1)      adds private key identities to the authentication agent...on the command line.
If any file requires a passphrase, ssh-add asks for the passphrase from the user. The passphrase
is read from the user's tty. ssh-add retries the last passphrase if multiple identity files are
given...

chpass(1)       add or change user database information
add or change user database information

useradd(8)      add a user to the system
The useradd utility adds a user to the system, creating and populating a home directory if
necessary. Any skeleton files will be provided for the new user if they exist in the skel-dir
directory (see the k option). Default...
.
.
.
```

Listing 7: Add new user

```
$ apropos “make directory”
make(1) maintain program dependencies
...CURDIR A path to the directory where make was executed. Refer to the description of PWD for
more details. MAKE The name that make was executed with argv[0]. For compatibility make also sets
.MAKE with the same value. The...

mkdir(1)        make directories
make directories

ln(1)   make links
...a directory in which to place the link; otherwise it is placed in the current directory If
only the directory is specified, the link will be made to the last component of source_file
. Given more than two arguments, ln makes...

mkfifo(1)       make fifos
make fifos...of a=rw mkfifo requires write permission in the parent directory. mkfifo exits 0 if
successful, and >0 if an...

mkdir(2)        make a directory file
...will contain the directory has been exhausted. EDQUOT The user's quota of inodes on the file
system on which the directory is being created has been exhausted. EIO An I/O error occurred
while making the directory entry or...
```

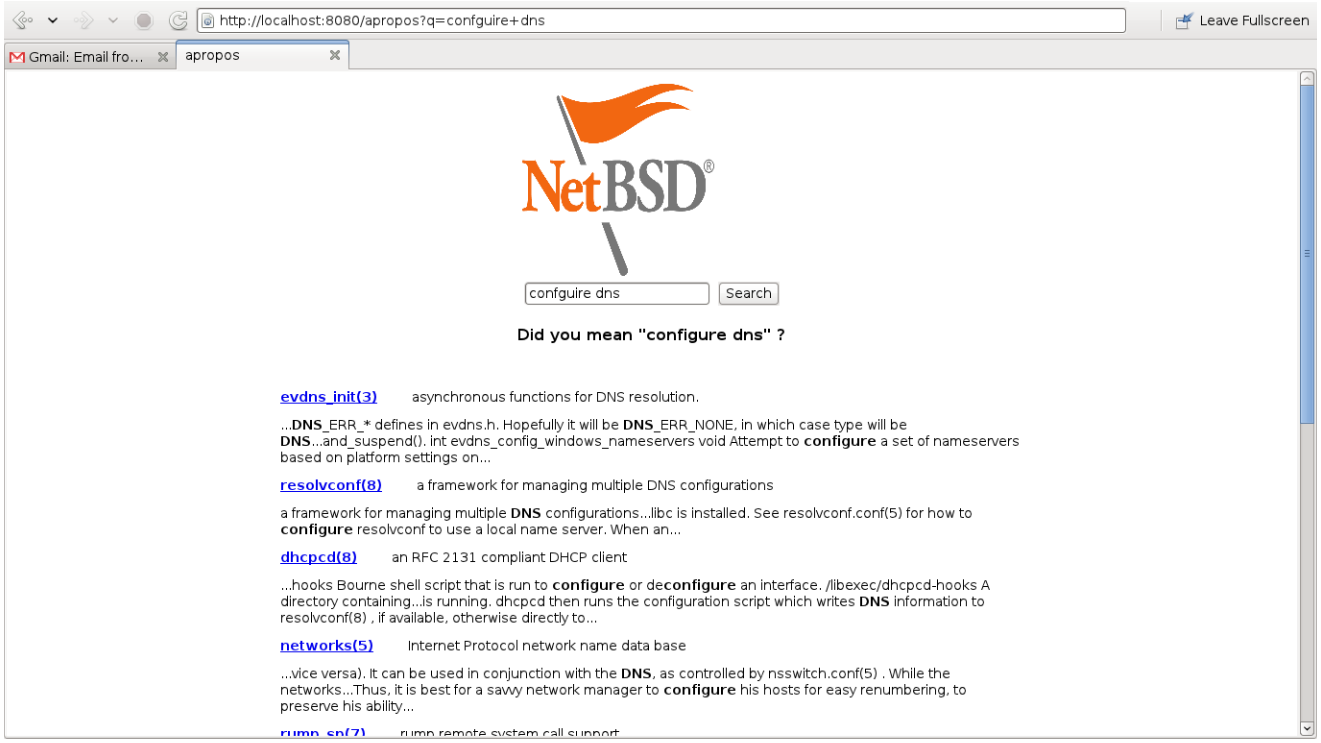
Listing 8: make directory

```
$ apropos “signal number to string”
psignal(3)      system signal messages
...the signal number is not recognized sigaction(2) , the string Unknown signal is produced. The
psiginfo function produces the same output as the psignal function, only it uses the signal
number information from the si argument. The message strings can...

intro(2)        introduction to system calls and error numbers
...undefined signal to a signal(3) or kill(2) function). 23 ENFILE Too many open files in system.
Maximum number...shell. Pathname A path name is a NUL -terminated character string starting with
an optional slash, followed by zero or...

dump_lfs(8)     filesystem backup
...low estimates of the number of blocks to write, the number of tapes it...a string containing
embedded formatting commands for strftime(3). The total formatted string is...in. If dump_lfs
rdump_lfs receives a SIGINFO signal (see the status argument of stty...
```

Listing 9: signal number to string



Did you mean "configure dns" ?

evdns_init(3) asynchronous functions for DNS resolution.

...**DNS**_ERR_* defines in evdns.h. Hopefully it will be **DNS**_ERR_NONE, in which case type will be **DNS**...and_suspend(). int evdns_config_windows_nameservers void Attempt to **configure** a set of nameservers based on platform settings on...

resolvconf(8) a framework for managing multiple DNS configurations

a framework for managing multiple **DNS** configurations...libc is installed. See resolvconf.conf(5) for how to **configure** resolvconf to use a local name server. When an...

dhcpcd(8) an RFC 2131 compliant DHCP client

...hooks Bourne shell script that is run to **configure** or de**configure** an interface. /libexec/dhcpcd-hooks A directory containing...is running. dhcpcd then runs the configuration script which writes **DNS** information to resolvconf(8) , if available, otherwise directly to...

networks(5) Internet Protocol network name data base

...vice versa). It can be used in conjunction with the **DNS**, as controlled by nsswitch.conf(5) . While the networks...Thus, it is best for a savvy network manager to **configure** his hosts for easy renumbering, to preserve his ability...

rump_sp(7) rump remote system call support

Figure 1: The spell corrector and the CGI frontend in action

8 Availability

The code for this project has been imported into the NetBSD source tree and being maintained there. Experimental work is still being done in the *github* repository of the project. It can be obtained from:

https://github.com/abhinav-upadhyay/apropos_replacement

9 Acknowledgement

This project has been developed as part of *Google Summer of Code 2011* [17], thanks to **Google** for sponsoring it. Special thanks to **Kristaps Dzonsons** who is the developer of the *mdocml* [18] project, he also helped by pointing out several issues in the parsing related code. Thanks to **David Young** who was involved with this project closely and offered useful guidance and encouragement throughout. A special thanks goes to **Thomas Klausner** who helped in writing and reviewing the man pages for this project. Thanks to **Petra Zeidler** for administering the GSoC program for *The NetBSD Foundation*.

References

[1] *NetBSD manual page for the old version of apropos* `http://netbsd.gw.com/cgi-bin/man-cgi?apropos++NetBSD-5.1`

[2] *NetBSD manual page for makewhatis(8)* `http://netbsd.gw.com/cgi-bin/man-cgi?makewhatis++NetBSD-5.1`

[3] Brin, S.; Page L. *The Anatomy of a Large-Scale Hypertextual Web Search Engine* Computer Networks and ISDN Systems 30:107-117, 1998

[4] Manning; Raghwan; Schutze *Introduction to information retrieval*, 3-9, 2008

[5] Porter, M. F. *An algorithm for suffix stripping*, Program, 14(3): 130-137, 1980

[6] *mdocml online manual page for libmandoc* `http://mdocml.bsd.lv/mandoc.3.html`

[7] *Sqlite home page* `http://sqlite.org`

[8] *Sqlite FTS3 and FTS4 Extensions* `http://sqlite.org/fts3.html`

[9] *Online manual page for makemandb(8)* `http://netbsd.gw.com/cgi-bin/man-cgi?makemandb++NetBSD-current`

[10] *Online manual page for apropos-utils(3)* `http://netbsd-soc.sf.net/projects/apropos_replacement/apropos-utils.html3`

[11] *man-db, the on-line manual database* `http://man-db.nongnu.org/`

[12] *Online manual page for mandocdb(8)* `http://mdocml.bsd.lv/mandocdb.8.html`

[13] *NetBSD online manual page for btree(3)* `http://netbsd.gw.com/cgi-bin/man-cgi?btree+3+NetBSD-5.1`

[14] Fuhr, Norbert *Probabilistic models in information retrieval* The Computer Journal 1992

[15] Zaragoza, H.; Craswell, N.; Taylor, M.; Saria, S.; Robertson, S. *Microsoft Cambridge at TREC13: Web and HARD tracks* In proceedings of TREC-2004

[16] *Online manual page for man(7)* `http://mdocml.bsd.lv/man.7.html`

[17] *Google Summer of Code home page* `http://code.google.com/soc/`

[18] *The mdocml project home page* `http://mdocml.bsd.lv`

[19] Jones, Sparck K. *A statistical interpretation of term specificity and its application in retrieval.* Journal of Documentation Volume 28 Number 1 1972 pp. 11-21

[20] Salton, Gerard; Buckley C. *Term-Weighting Approaches in Automatic Text Retrieval.* Information Processing & Management Vol. 24, No. 5, pp. 513-523, 1988

[21] *Online manual page for mdoc(7)* `http://mdocml.bsd.lv/mdoc.7.html`

Notes

[1] Although the new *apropos* by default returns 10 results for a query but in the listings the output has been snipped to save space