Three full-text desktop search engines

Needle in a Haystack

Desktop search engines such as Tracker, DocFetcher, and Recoll help track down files by their content, even in massive datasets. By Harald Jele

Some people say that keeping things tidy just means you're too lazy to search. However filesystems are not fixed, not necessarily logical or self-explanatory, and can change over time. Even for the tidiest of computer aficionados, it can be helpful and indeed essential to use search functions to find what was once stored, even into the furthest corners of a deeply nested storage system. This capacity is especially important if you want to search through a large volume of files, the content of which you are not familiar. For this kind of use case, it makes sense to take a closer look at the search functions on desktop computers and their possibilities.

The event that impelled me to author this article was the arrival of a 13GB bundle of compressed files, the contents of which could possibly be helpful in my research. To find out, I had to browse through the flood of data, aided only by standard search functions. Manual browsing, searching, and quick reading would have been too prone to error on the one hand and too time-consuming on the other. Thoroughly sifting through 554 files – each the size of an average daily newspaper – with trained eagle eyes would have used up some of my remaining lifetime and possibly only returned mediocre results.

The obvious approach was to test the suitability of the desktop's built-in mechanisms for full-text search. In the present case, a system with Ubuntu 20.04 LTS and a Gnome interface formed the basis of the default installation. As a first basis for the search, the inconspicuous but quite powerful Tracker [1] program was investigated. An Internet search revealed at least two other recent tools that, according to their brief descriptions, would be suitable for the task I set: DocFetcher [2] and Recoll [3] specialize in full-text search and were built for use on a modern desktop. On server systems, the combination of Solr and Lucene [4] is considered the standard for implementing an indexing system for full-text searching and making the results accessible by means of a search engine. The duo shows how powerful modern search systems can be. Today's PCs and recent laptops offer enough performance to index files with this combination; however, the high overhead is hardly reasonable for average desktop users and is clearly over-the-top if you consider the usual requirements when working on a PC. Nevertheless, many common applications for the desktop are oriented toward the performance characteristics of Solr and Lucene.

The Regain [5] project was another product that used Lucene as a search engine on the desktop. However, it was discontinued after the release of version 2.1.0 in 2014.

Tracker

If you want to come to grips with Tracker, you'll first have to embark on a lengthy search of another kind. Although the software is maintained within the Gnome project, the documentation from the two main developers, Sam Thursfield and Carlos Garnacho, leaves much to be desired. In some parts of the docs you will find outdated information from older versions, with announcements for future enhancements that were never implemented. Interested parties are largely left to their own devices when trying to determine Tracker's current feature set. The blog [6] maintained by Sam Thursfield is interesting and instructive. He offers readers detailed information about the decisions that ultimately had to be made during development.

Photo by Lucas Gallone on Unsplash

Tracker essentially comprises two parts: a SPARQL database built around SQLite and what are known as "tracker miners." The SPARQL graph-based query language was defined by the World Wide Web Consortium (W3C) and has been available as a stable version since March 2013 [7]. The tracker miners, which are implemented as classic daemons, browse specified file paths and prepare the data found for indexing.

Tracker was developed from the beginning as an application intended to go efficiently about its work in the background wherever possible without causing a stir. The developers also set store on indexing not slowing down the usual desktop work to any great extent. Moreover, they wanted to avoid a power-hungry indexing tool draining laptop batteries and leaving the user blissfully unaware of the reason. Tracker is modular, not monolithic, which makes the application very flexible but also a bit confusing, in turn extending the learning curve.

On the Ubuntu desktop, you select the paths and file types that will end up in the index in *Settings* | *Search* | *Search Locations* and choose from the *Places*, *Bookmarks*, and *Other* tabs. In the terminal, you then need to stop and restart the tracker daemon to apply the changes to the configuration. The associated commands, and the most important commands for improved control of the work in progress, are summarized in Table 1. A complete overview of the `tracker` tool parameters is provided in the tracker command-line interface (CLI) documentation [8]. (Note that the most recent version of the tool is `tracker3`.)

Tracker keeps its journal in the logged-in user's directory under `~/.local/share/tracker/data/tracker-store.journal`. Changes to the filesystem will have an effect. If the journal does not change, it also means that the tracker processes have completed all pending work and that all data is covered by the full-text index.

A search engine should distribute the time-consuming process of indexing across many processes, but Tracker does not do this. Regardless of how many files need to be processed and how many CPUs the computer has available, only two indexing processes are active at any given time.

Many of the tasks involved in working with Tracker can be completed either from the command line or with the help of desktop tools. For example, if you open the default file manager, you can use its search function to extend the search to file content. As Figure 1 shows, you can switch between *File Name* and *Full Text* while searching and set all kinds of restrictions.

If you select *Full Text*, Tracker applies the search term ("kernel" in Figure 1) to the contents of selected files and looks up the term in the full-text index. For hits, the file manager displays the associated files, as well

Table 1: Important Tracker Commands

`tracker daemon -s`	Start the daemon and its processes.
`tracker daemon -t`	Stop the daemon and its processes.
`tracker daemon --watch`	Show what Tracker is currently processing.
`tracker daemon --set-log-verbosity`	Set the verbosity of the daemon.
`tracker status`	Show the status of the current indexing process.

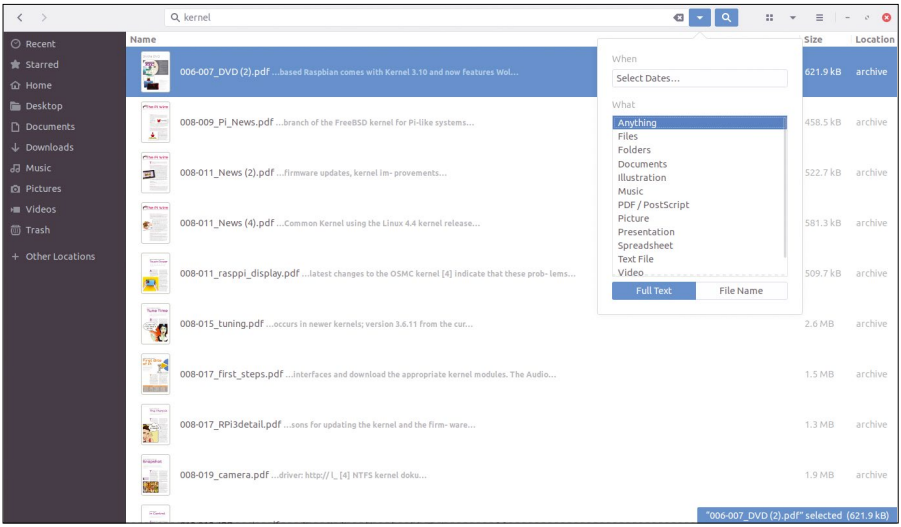

Figure 1: An example of a Tracker full-text search in the Gnome desktop file manager.

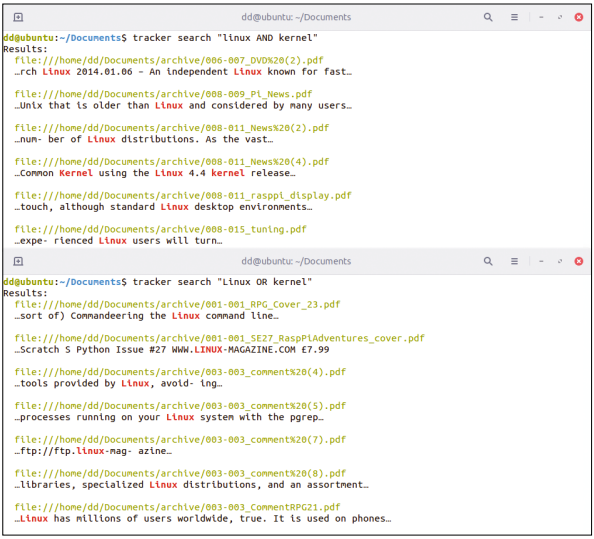

Figure 2: The terms AND, OR, and NOT are used for logical linking when searching in Tracker.

as a short preview of the context in which they were found. If you want to search for two or more terms at the same time, you won't find them with the file manager – not because Tracker can't do that but because that is simply not implemented in the file manager. Even the *Documents* option, which looks to be a kind of document manager, does not currently implement this capability, nor does it show you a preview of the discovered terms or offer to highlight the discovered terms in the document. However, Tracker provides both pieces of information.

In the terminal, a simultaneous search for two or more terms works in the expected way with the corresponding logical operators (AND/OR/NOT), as seen in Figure 2. Tracker deliberately does not go beyond these operators into logical linking. Thursfield writes in his blog that average users wouldn't use other logical links even if they were available. Among other things, he refers to proximity operators such as NEAR from information retrieval, which probably only a few experts use in a classic full-text search. The same applies to word stemming, which Thursfield discusses in the blog, but which Tracker ultimately does not implement.

Tracker fulfills many of the requirements for a semantic desktop. When mapping the terms in the index, the daemon also stores those text elements from which keywords originate, allowing you to specify the text or metadata element in which the match must occur.

The database maps this by defining an ontology. By default, Tracker uses the variant popularized by the Nepomuk [9] project funded by the European Union between 2006 and 2008. The ontology is not hard-coded in Tracker and can be replaced by any other ontology, if needed, or independently extended and modified. The Tracker Ontology Reference Manual [10] gives a good overview of the Nepomuk elements.

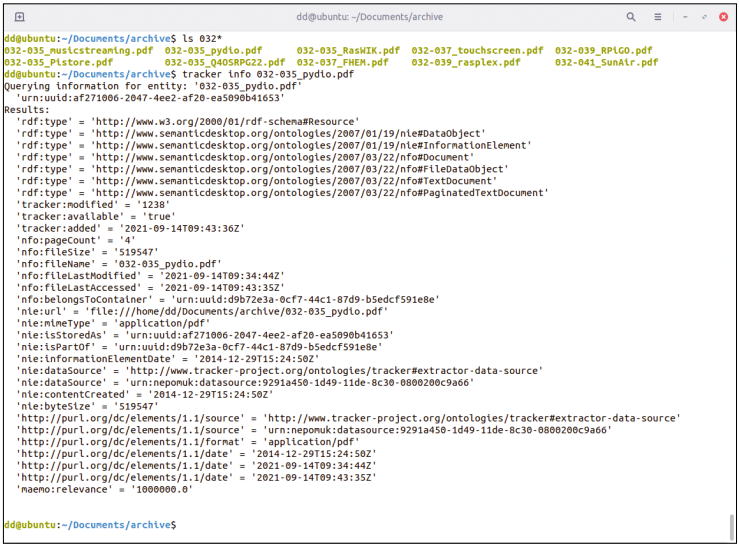

Figure 3: Elements of the Nepomuk ontology used by Tracker, as displayed in the terminal.

The command

```
tracker info <file>
```

lets you analyze individual files, in advance of indexing, for their compliance with the deployed ontology rules. Figure 3 shows part of the corresponding output in the terminal. Tracker needed 7:05 minutes to index my 554 files with a total size of 13GB on my setup, which is pretty good compared with the other two candidates. That said, the three candidates do not all have the same feature set.

DocFetcher

DocFetcher, also an open source program for full-text searching on the desktop, has completely different requirements from Tracker. Its mission is to index predefined file paths as quickly and efficiently as possible at the push of a button. DocFetcher grabs the resources it needs without retiring unobtrusively to the background. Luckily, it does not completely block all other work on an average PC. However, with the requirements DocFetcher has during the install, it plants a significantly larger footprint on the computer than Tracker.

DocFetcher is available for Linux, Windows, and macOS and comes in two variants: the non-commercial DocFetcher and DocFetcher Pro (a test version of which was released in January 2021), which has undergone a complete overhaul compared with the non-commercial version. The commercial version is not limited in terms of function in the test version, but it is limited in terms of the display. For example, it only displays five results of a search instead of all of them; this is sufficient for an evaluation, say the developers. On the DocFetcher Pro website [11], the developers list the other differences between the commercial and non-commercial versions in detail.

On my system, I used version 1.1.22 of DocFetcher, which is available as a Snap package for Ubuntu and requires a Java installation:

```
$ sudo apt install default-jre
$ sudo snap install docfetcher
```

The monolithic structure of the program prevents the use of individual modules but allows a uniform view of the implemented search methods and operation. An up-to-date description, help pages, tips and tricks, and a user forum can be found on the project's website.

DocFetcher structures its display in frames (Figure 4) and does without a classic menu, which seems confusing at first. However, once you get used to right-clicking to call up the commands, the workflow is friendly and focused on the essentials.

In the lower left frame (*Search Scope*), you specify the file paths you want DocFetcher to index. In the *Document Types* frame above, you specify the file types to be indexed and limit their file size, if necessary. At top right, a bar lets you enter search terms. Below that, DocFetcher lists the files in which at least one match occurred for the search term. If you select one of the lines, a preview of the corresponding file appears in the bottom right window, along with the color-highlighted locations where the match was found.

By default, DocFetcher logical ORs the terms entered in the search bar, instead of using a logical AND, as is common with many other search engines. The AND, OR, and NOT logical operators are available. If you are searching for a phrase in which several terms must occur in the order entered, you need to quote the search terms.

DocFetcher also has a proximity search option that works when you append the proximity operator (a tilde) to a phrase. For example, entering *"Bludenz Bregenz"~10* causes the tool to rank texts in which the two names of these Austrian cities occur no more than 10 words apart as matches. If you do not specify a value, DocFetcher assumes a distance of zero and searches for the two juxtaposed terms. Ten words is the maximum distance the search engine accepts.

The tool also can handle some very specialized search options. Boosting lets you give a higher weighting to individual search terms. In a field search, it searches for terms in the

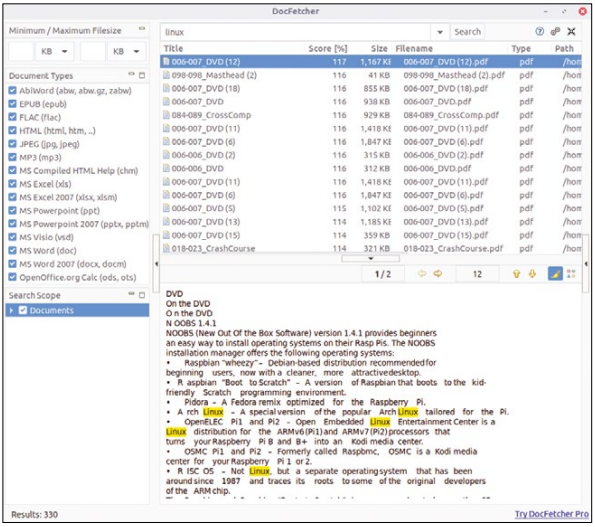

Figure 4: The display in DocFetcher. A classic menu is missing; operations are largely triggered by mouse clicks.

filename, *title*, and *author* fields in documents with metadata. The range search lets you find terms that are within a defined lexicographic range.

Apart from text files, DocFetcher can include email documents in its index but not graphics or multimedia files, which, along with compressed files, are reserved for the commercial variant.

If an index has been created but the associated files have changed in the meantime, DocFetcher does not automatically re-index them. A separate daemon detects and logs changes. Its records can be applied when you are reorganizing the overall index to process only those files and directories that have changed in the meantime. Figure 5 shows some of the options you can control from the menu in the search area.

Some users might be interested to learn that a portable version of DocFetcher is also available, which makes it possible to take the program and associated files with you (for example, on a mobile data carrier) and run the program on that device. In the same way, DocFetcher and the indexed documents can be transferred from one computer to another without having to re-index the data. Switching between the non-commercial and the commercial edition of DocFetcher annoyingly forces you to re-index your documents completely because the two versions (currently) cannot work with the same index files. When building the index, special attention should be paid to files that are not UTF-8 encoded: DocFetcher does not index them correctly per se.

It took DocFetcher just under 15 minutes to index the 554 PDF files from the test suite of 13GB.

Recoll

With Recoll, the leader in desktop search programs enters the fray. The Ubuntu repository offered version 1.26.3 at the time of writing this article. The Personal Package Archive (PPA) maintained by the developers had the latest version at that time, v1.30.1. A Snap package was not available. I tested the version from the PPA, which is installed with the commands:

```
$ sudo add-apt-repository ↵
  ppa:recoll-backports/recoll-1.15-on
$ sudo apt-get update
$ sudo apt-get install recoll
```

Versions of Recoll are available for Linux, a number of Unix variants, Android, Windows, macOS, and even OS/2. Its high performance as a versatile desktop tool comes from the use of the Xapian [12] search engine, which does the real heavy lifting in the background. Xapian's feature list is endless, and Recoll implements most of it.

Essentially, the connection to the search engine is implemented by a variety of Python scripts. Xapian, and thus Recoll, is designed predominantly for full-text searching. Indexing non-text files takes a bit of a back seat, although Xapian also includes the metadata of multimedia and graphics files in the index.

Like DocFetcher, Recoll assumes that text is UTF-8 encoded by default and trips up over files that deviate from the norm. That said, Recoll's mandatory filter files are equipped to handle a large range of encoding types.

When first launched, Recoll asks whether you want to set up the directories with the content you want to index right away or postpone this step until a later time. If you choose to index immediately, Recoll confronts you with several options (Figure 6). You will want to focus mainly on word stemming (reduction to the root lexeme) and choose the languages to be used. Also, go to the *Unac exceptions* field and define the characters that Recoll should take into account when indexing. Recoll will ignore all others, such as special characters, as well as combinations of basic Latin letters and diacriticals.

The best strategy is to automate Recoll's index runs in a cron job so that new or changed data is indexed on a regular basis. No daemon monitors the filesystems for changes. Unlike the other two services, Recoll got to work immediately, with five indexing jobs quickly completed. On

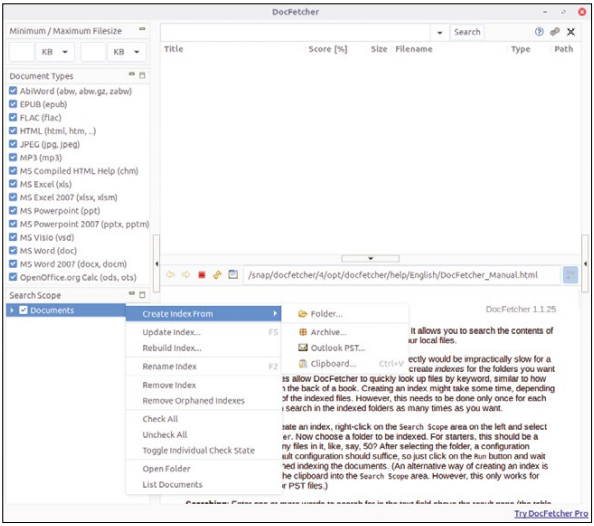

Figure 5: DocFetcher's index can be reorganized quickly with the help of a daemon that detects and logs changes.

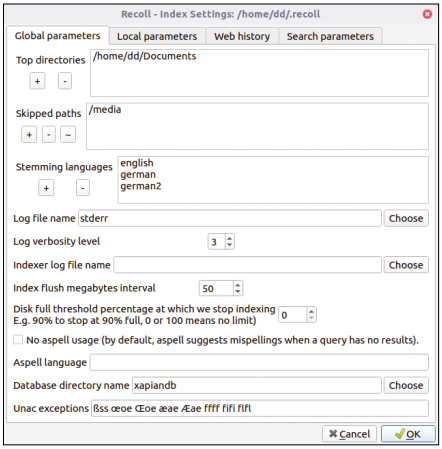

Figure 6: Initial settings for indexing in Recoll.

my setup, the 554 PDF files were indexed and searchable after just under 2:30 minutes – quite a significant difference compared with the almost 15 minutes that DocFetcher needed for the same task.

After completing the index run, Recoll reports whether it was able to index all data or had to omit certain files (e.g., because of missing utilities). If you retroactively install the missing programs, the skipped files can then be indexed. Recoll impresses with an exemplary display of the search results. Figure 7 displays the match list, which provides a minimal preview of some match locations. Clicking on the *Snippets* link shows you all match locations in a document. Thanks to these options, you can very quickly access the results that are particularly good matches for the query. More than just the search and index parameters can be customized in-depth in Recoll. The user settings bring even more options that let you control how the interface and match lists are displayed or that create links to external index files.

The query language display window (Figure 8), which always pops up when the cursor comes to a stop in the input line while you are working on a search query, proves to be particularly helpful. Recoll tells you which shortcut operators can be used and gives you examples of how to use each of them. Although this tutorial might not be necessary for Boolean operators, it definitely helps for other operators, especially if you happen to use several search engines at the same time that work in similar ways but differ in the details.

Under Linux, Recoll can be integrated as the default desktop search engine if so desired. The Gnome *Recoll Search Provider* plugin takes effect for all search actions and returns the results from the Recoll index. This addition can make life far easier for users who manage large volumes of text on the desktop and constantly need to search for specific terms.

Additionally, Recoll lets you maintain "facets," with which you can create meaningful subsets of particularly large match lists. Facets can mean the media type (text, image, video, email, etc.), the file creation time, or the last change date. Before you get too excited, though, faceting is limited to these predefined criteria.

Conclusions

An alternative full-text search tool on the Linux desktop quickly returns dividends if you work frequently with very large directories and a large number of files. The capabilities of the search engines presented here (Table 2) are usually fit for the purpose but not always easily

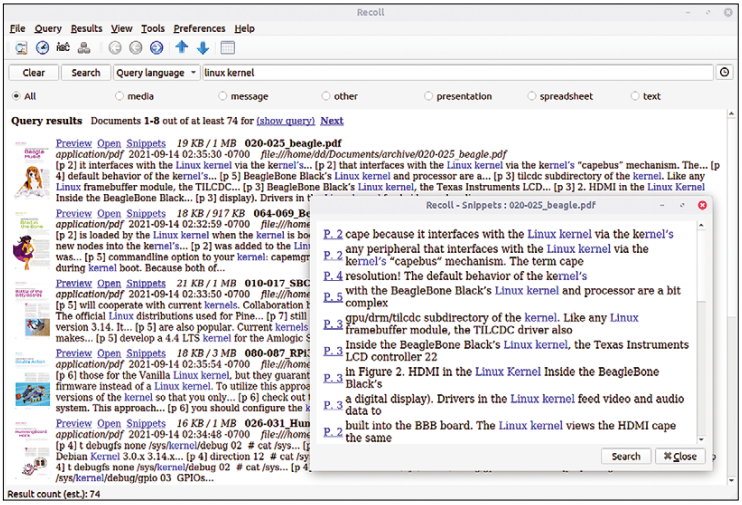

Figure 7: Recoll provides a short display of the results (Snippets) and match locations in the document.

accessible or user friendly. This is especially true for the otherwise powerful Gnome standard tool, Tracker, whose feature set is not fully utilized by any of the associated desktop applications.

The test shows that both DocFetcher and Recoll cut a fine figure on the Gnome desktop and meet upscale requirements. Both also have unique selling points that could tempt some users. In the case of DocFetcher, this might be the mobile version, and in the case of Recoll, the complex indexing and faceting options.

Although search engines in the past often came with complex search masks and languages, today they are usually content to show a simple input line and a very low-key query language. This trend is also propagating onto the desktop. Earlier approaches are now more likely to give way to a sensible sorting of (large) results sets, as well as the possibility of subsequently breaking these sets down into increasingly smaller sets by applying smart faceting choices, to ultimately generate useful match results without wasting time. ■

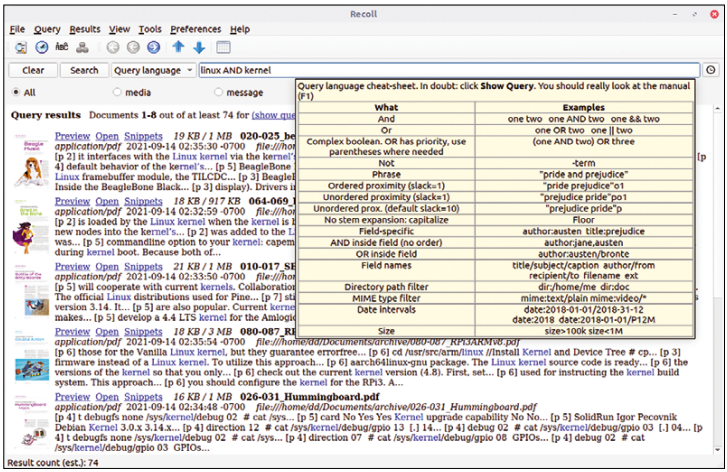

Figure 8: Help for the query language in Recoll shows numerous useful hints.

Table 2: Search Engine Features

Feature	Tracker	DocFetcher	Recoll
Character encoding detection	+	–	–
Search term highlighting	+	+	+
Boolean join operators	+	+	+
Proximity operators	–	+	+
Multilingual word stemming	–	–	+
Faceting search results	–	–	+
Indexing multimedia files	+	–	+
Bindings/open APIs	+	–	+
Weighting of search terms	–	+	+
Phrase search	–	+	+
Synonym searching	–	–	+
Mobile version	–	+	–
Indexing SQL databases	–	–	+
Desktop integration	+	–	+
Support for multiple operating systems	–	+	+
Wildcard (placeholder) searching	–	+	+
Autocompleting search queries	–	–	+
Time (min) to index 554 PDFs (13GB)	7:05	14:40	2:30
No. of indexing processes	2	1	5

Info

[1] Tracker: [https://gnome.pages.gitlab.gnome.org/tracker/]

[2] DocFetcher: [https://sourceforge.net/projects/docfetcher/]

[3] Recoll: [https://www.lesbonscomptes.com/recoll/]

[4] Solr: [https://solr.apache.org]

[5] Regain: [http://regain.sourceforge.net]

[6] "Tracker 3.0: Where do we go from here?" by Sam Thursfield: [https://samthursfield.wordpress.com/2020/11/05/tracker-3-0-where-do-we-go-from-here/]

[7] SPARQL 1.1 Overview: [https://www.w3.org/TR/sparql11-overview/]

[8] Tracker CLI documentation: [https://gnome.pages.gitlab.gnome.org/tracker/docs/commandline/]

[9] Nepomuk: [https://nepomuk.semanticdesktop.org]

[10] Tracker Ontology Reference Manual: [https://developer-old.gnome.org/ontology/stable/]

[11] DocFetcher Pro: [https://docfetcherpro.com/features/]

[12] Xapian: [https://xapian.org]

Author

Harald Jele is a staff member at the University of Klagenfurt in Austria. In 1993 he came across Linux by a happy coincidence, and he has been using it on servers and desktops ever since.