Tagsistant

All your files, quickly and easily.

Tagsistant 0.8 howto

Last Updated on Sunday, 23 November 2014 11:45

v. 0.8

0.8 is the new release of Tagsistant, a semantic filesystem for Linux kernels. Release 0.8 is still under development and can be downloaded from the SVN trunk (more information here).

A semantic filesystem is a new kind of filesystem for personal usage that organizes contents by tags. Tagging our contents is a common practice today. We use tags on-line for photos, bookmarks and much more. Social networks let tag near everything. Our media library is full of tag (EXIF in JPEG, ID3 in mp3, ...) and html or other documents contain keywords which are basically tags.

Despite all this load of tags (metadata, keywords, call it as you prefer), the desktop is still an ancient kingdom ruled by directories. We are forced to use directory hierarchies to store our contents and often we struggle to locate our files. Many tools have been introduced to port the concept of tags to the desktop, but all of them are applications which store tags inside a proprietary or somehow closed database that forces you to use their interface to take advantage of tags. This has some consequences:

1. tags must be entered and deleted using that application interface
2. tags must be used from that application interface when querying for contents
3. contents (files) must be modified using that application interface (does not apply to file managers only)
4. no exchange with other tagging tools is permitted (two applications not necessary are able to share tags)
5. tags can't be accessed by an external program or service (like a web service or an indexing tool)

Tagsistant is an effort to solve all those problems by moving the tagging logic from the application layer to the storage layer. If tags are managed by the filesystem, all the software you already use will be able to access and take advantage of tags: create tags, delete tags, tag files, retrieve files!

The first released Tagsistant version was 0.2, in 2007. Back then Tagsistant allowed the tagging of files only. Starting from release 0.6, Tagsistant started to manage any kind of object, including devices, directories and so on. Some syntactic sugar has been added to accomplish this, like the @/ directory at the end of the queries.

But release 0.8 adds a lot more. A deduplication layer reduces disk usage by deleting file duplicates preserving tagging information. An internal caching layer speeds up the execution of queries already resolved. Tags have been expanded and now support triple form, better known as

machine tags. Frequently used queries can be bookmarked as aliases, saving some user time. Tags can be organized with relations to form a simple kind of ontology. And Tagsistant does even more to make the user life easier, by automatically extracting tags on its own, using its customizable autotagging plugins.

To start using Tagsistant, you need to mount it somewhere. In this tutorial we'll choose by convention the ~/myfiles directory, but you can change it as it pleases you:

```
$ tagsistant ~/myfiles
 Tagsistant (tagfs) v.0.8 Build: 20130323.000045 FUSE_USE_VERSION: 26
 (c) 2006-2013 Tx0
 For license informations, see ./tagsistant -h

 Using default repository /home/tx0/.tagsistant
 Using default plugin dir: /usr/local/lib/
```

By default Tagsistant saves all its managed informations (files, tags, relations and tagging status) in a hidden directory named ~/.tagsistant. If you use just one Tagsistant repository, just ignore this information. But if you plan to use more than one Tagsistant repository at the same time, please remember to provide a separate one to each filesystem, using the --repository argument, as in:

```
$ tagsistant --repository=~/.photo ~/myphoto
$ tagsistant --repository=~/.music ~/mymusic
```

The repository can also be passed as the first parameter without the --repository= prefix. This allows you to mount Tagsistant from /etc/fstab using the special tagsistant# prefix before the repository path. This is an example line:

```
tagsistant#/home/tx0/.tagsistant /home/tx0/myfiles   fuse   noatime,user,uid=1000,gid=1000   0   0
```

You can of course set up as many entries you need in separate directories:

```
tagsistant#/home/tx0/.tagsistant-photos /home/tx0/photos   fuse   noatime,user,uid=1000,gid=1000   0   0
tagsistant#/home/tx0/.tagsistant-music  /home/tx0/music    fuse   noatime,user,uid=1000,gid=1000   0   0
tagsistant#/home/tx0/.tagsistant-docs   /home/tx0/docs     fuse   noatime,user,uid=1000,gid=1000   0   0
```

Add allow_other if you want the filesystem to be accessed by other users (do this for exporting tagsistant to Windows via Samba). Add noauto to avoid automounting at boot time. If mounted at boot time, Tagsistant will run as root and only root will be able to unmount it.

Another thing Tagsistant does by default is using SQLite. If you feel comfortable with SQLite or just don't know what does it mean, feel free to skip the rest of this section. If instead you would prefer to use MySQL, change the command line as follows:

```
$ tagsistant --db=mysql:host:database:user:password ~/myfiles
```

Of course you must provide the database and the user inside MySQL *before* mounting Tagsistant.

You can omit the tokens after mysql, accepting default values, but if you specify a token you must specify the tokens on its left too. So, if you just write --db=mysql, you are using default values of *localhost, tagsistant, tagsistant* and *tagsistant* for the other tokens.

This schema gives you the flexibility to maintain just one standard *tagsistant* user inside MySQL with password *tagsistant*, but allowed to access many tagsistant databases. To connect to DB *photos* or *music*, you'll just change the db name like in: --db=mysql:localhost:photos or --db=mysql:localhost:music.

Now Tagsistant is managing the ~/myfiles directory. If you list its contents you'll find something similar to:

```
$ ls -l ~/myfiles
total 1.6M
drwxr-xr-x 2 tx0 tx0 400K Mar 23 16:40 alias
drwxr-xr-x 2 tx0 tx0 400K Mar 23 16:40 archive
drwxr-xr-x 2 tx0 tx0 400K Mar 23 16:40 relations
drwxr-xr-x 2 tx0 tx0 400K Mar 23 16:40 stats
drwxr-xr-x 2 tx0 tx0 400K Mar 23 16:40 store
drwxr-xr-x 2 tx0 tx0 400K Mar 23 16:40 tags
```

The main directories archive/, alias/, store/, relations/, stats/ and tags/ are the interface you'll use to interact with Tagsistant. It's very important that you understand the meaning Tagsistant gives to each of them, since it's different and can be unexpected compared to the experience you have with traditional filesystem.

To skim over the role of each directory, we'll briefly describe each of them in a rational order. The tags/ directory is devoted to tag management. Inside it you can create, rename and delete tags. This can also be done in the store/ directory, but be warned that deleting a tag here can wipe out your repository. Do it in tags/ instead. Use store/ to tag your files, to perform queries and to access your contents from your favorite application (media player, text editor, drawing app, ... whatever!). When a file is created under store/ it appears inside archive/ too. Here you can edit it, but you can't delete it, or create other files. You can even access all your files in one single directory. The relations/ directory contains the relations between your tags to organize your knowledge. In alias/ you can save your most used queries while in stats/ you can get information about Tagsistant internal state or how it was started. Use the chart below to summarize the role of each one.

Directory	Manage tags	Manage files	Manage relations	Manage bookmarks	Stats&conf
alias/				✓	
archive/		✓ **[1]**			
relations/			✓		
stats/					✓
store/	✓ **[2]**	✓			
tags/	✓				

Notes:

1. Files can be edited, but can't be created or deleted. Use store/ instead.
2. Deleting tags can wipe out your whole repository! Use tags/ instead.

There are two places where you can create tags: the store/ directory and the tags/ directory.

The tags/ directory is the recommended place to manage your tags. You can create them, rename them and delete them too. But the tags/ directory does not allow you to tag your files. This role is assigned to the store/ directory. There you can create and delete tags too, but you can also tag files by copying them inside one or more tags, but is not advisable to use the store/ directory do *delete* tags. We'll return on this and on the tags/ directory later. So far just assume you'll be using the store/ directory only.

In respect to an ordinary directory, the store/ directory is something completely new. Be prepared to learn something before using it. Inside store/ the assumption that two directories can't be parent of the other at the same time falls. But let's start from the most simple operation: creating a tag. Remember that in Tagsistant a tag is just a directory created right under the store/ or tags/ directories. Knowing this, all we have to do is to use mkdir:

```
$ mkdir ~/myfiles/store/video/
$ mkdir ~/myfiles/store/scifi/
$ mkdir ~/myfiles/store/startrek
```

What we have done here can be translated in English as: create a tag called *video*, a tag called *scifi*, and a tag called *startrek*.

To understand the binding between store/ and tags/, list the contents of tags/ and you'll find the tags you've just created under store/:

```
$ ls ~/myfiles/tags/
video scifi startrek
```

Now we leave the store/ directory to take a quick tour of another one, strictly related: the relations/ directory. This one is used (guess what?) to manage relations between tags. Relations keep clean and effective your knowledge about tags and reduce the number of times you have to tag files, like a virtual tagging mechanism. Let's see how.

Tagsistant is a semantic filesystem. The adjective semantic refers to the meaning of tags which inherently establishes relations between them. For example, a relation of *inclusion* inherently exists between *the concept of music* and *the concepts of rock and jazz*. Another example is represented by the tags we created previously: *scifi* and *startrek*. Those tags obviously are related because Star Trek is a sci-fi TV show.

Tagsistant provides a way to describe such relations in a way that can be computationally used. In Tagsistant any relation always involves two tags and can be of two types:

1. *left-tag is equivalent to right-tag (use 'is_equivalent')*
2. *left-tag includes right-tag (use 'includes')*

For example, you can tell Tagsistant that scifi/ includes startrek/ by using mkdir under relations/:

```
$ mkdir ~/myfiles/relations/scifi/includes/startrek/
```

That's it. Now all the files tagged as startrek are *inherently tagged* as scifi too. The concept of inherently tagged means that a file tagged as startrek will act as if it was tagged as scifi too, even if the files is not, and it will act like this as long as the relation between the tags *scifi* and *startrek* is established.

The left tag (*scifi* in our example) must already exist, but Tagsistant can create the right tag for you, like in:

```
$ mkdir ~/myfiles/relations/scifi/includes/starwars/
```

The *starwars* tag did not exist yet, but if you list tags/, you'll find the *starwars* tag too:

```
$ ls ~/myfiles/tags/
scifi   startrek   starwars   video
```

You noticed it? This is a small violation of what we have said before: tags can be created inside store/ and tags/ only. The auto-creation of the right tag under relations/ is just an aid to speed up your experience. For clearness, keep assumed that you can create tags inside store/ or tags/ only.

Let's wrap up what we have seen so far. We have created some tags (*video, scifi, startrek* and lastly *starwars*) and we have established two relations: *scifi includes startrek* and *scifi includes starwars*. Now we are ready to tag our files.

Tagging a file happens when we copy that file under the store/ directory. The path we copy the file in tells which tags are applied to the file. More than one tag can be applied to a file at the same time and the same file can be tagged twice or more.

As a first test, we use the movie "First Contact" and tag it as *startrek* by copying it inside the proper tag. The command is:

```
$ cp first_contact.avi ~/myfiles/store/startrek/@/
```

Split the path in its logical components to understand what is happening. We have:

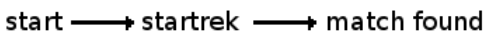
1. ~/myfiles: this is the mountpoint
2. store/: this clearly says we are tagging something
3. startrek/: this is the list (a one element list) of tags we are applying to the file. This is also called the *query part*.
4. @/: a conventional marker to end the query

Now let's check our file is where we put it:

```
$ ls ~/myfiles/store/startrek/@/
first_contact.avi
```

The @/ element is always used at the end of the tag list (the query) when we are asking Tagsistant to locate a file or to tag a file. Without the @/ mark, Tagsistant will assume that we are *still building the query*. A query without a @/ mark is called an incomplete query and can't be processed for looking up files.

This is a schema of the query:



To make a little more meaningful the role of @/, we'll now use the file we have just tagged with a media player:

```
$ mplayer ~/myfiles/store/startrek/@/first_contact.avi
```

Try to imagine how the path would look without the @/ mark. How could Tagsistant know that first_contact.avi is a filename and not another tag? It just can't. That's why a query must be completed by @/ to locate a file.

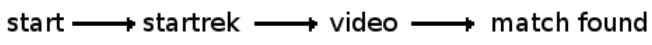
In the next example we use more than one tag:

```
$ cp the_wrath_of_khan.avi ~/myfiles/store/startrek/video/@
```

Translated in English this sounds like *tag the movie "The Wrath of Khan" as both 'startrek' and 'video'*. In this query the parts of our list are:

1. ~/myfiles: this is the mountpoint
2. store/: this clearly says we are tagging something
3. startrek/video/: two tags are being applied to the same file at once
4. @/: a conventional marker to end the query

And this is the query schema:



You will now find the file "the_wrath_of_khan.avi" inside store/video/@/, store/startrek/@/, store/video/startrek/@/ and store/startrek/video/@/. The last two queries are totally equivalent. But wait! We taught Tagsistant that scifi/ includes startrek/, so we expect to see that file in store/scifi/@/ too:

```
$ ls ~/myfiles/store/scifi/@/
first_contact.avi the_wrath_of_khan.avi
```

Yes, both startrek movies are there! That's because Tagsistant has an internal reasoner which uses the relations you provide to include files not directly tagged (remember? this is what we called *inherent tagging*). Now let's tag something else:

```
$ cp the_empire_strikes_back.avi ~/myfiles/store/starwars/@/
$ ls ~/myfiles/store/scifi/@/
first_contact.avi  the_wrath_of_khan.avi  the_empire_strikes_back.avi
```

No files are directly tagged as scifi/ but since that tag includes both startrek/ and starwars/ now it features three files. The real benefits of using relations are:

1. reducing the length of your queries
2. avoid re-tagging and over-tagging your files

Both save your time.

Another way to get the same set of results from the previous query would be searching all the files tagged startrek/ and all the files tagged starwars/, with this query:

```
$ ls ~/myfiles/store/startrek/+/starwars/@/
first_contact.avi  the_wrath_of_khan.avi  the_empire_strikes_back.avi
```

As you can see, the query scifi/@/ is much shorter than the query startrek/+/starwars/@/ but that's not the point.

The new mysterious mark +/ we've just introduced in the middle of the path is a way to build complex queries. It splits the query in two or more parts and processes them as separate queries. Then it merges the results of both queries to form one single superset of results to be returned to the user. To understand a query featuring one or more +/, first split the query by +/. You'll get two or more traditional sub-queries. Tagsistant will perform each sub-query separately and then it will merge the results just before returning them to you.

Translated to English, the previous query sounds like: get the results of the query store/startrek/@/ then merge them with the results of the query store/starwars/@/ and give me back what you've found.

This is totally different from writing two tags one next to the other, without a +/ in between. In that case you would be looking for files that are tagged by *both* tags, like in:

```
$ ls ~/myfiles/store/startrek/starwars/@/
```

which ends in no results (well, this could be no longer true since J.J. Abrams has got in charge of... but I'm straying from the topic).

So, you may be wondering: if I want to lookup all the videos and pictures of startrek from my collection, may I accomplish this with:

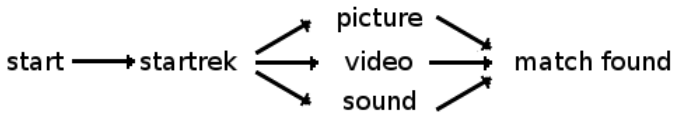
```
$ ls ~/myfiles/store/startrek/picture/+/startrek/video/@/
```

Yes, that's perfectly fine! A little bit long to write, but fine. And what about the teleport or red alert sounds you've saved as .wav files from the nineties? Just add them!

```
$ ls ~/myfiles/store/startrek/picture/+/startrek/video/+/startrek/sound/@/
```

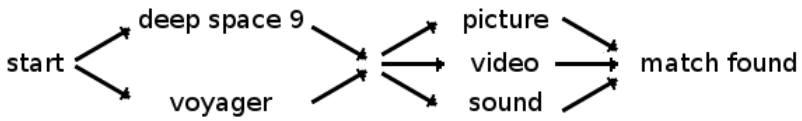
OK, that's definitely uncomfortable, I agree. But I've an alternative for you: a tag group! Tag groups are tag sets where one tag is enough to have a match. Your previous query is looking for all the files tagged as startrek/ and picture/ or video/ or sound/. The startrek/ tag is required, but any one of the other three tags would be enough. To exploit the compactness of tag groups, just include the last three tags between curly braces, like this:

```
$ ls ~/myfiles/store/startrek/{/picture/video/sound/}/@
```



Much shorter and more readable, isn't it? Tag groups can even be combined. Let's suppose you just want files related to Deep Space 9 or Voyager. Then you could query Tagsistant like this:

```
$ ls ~/myfiles/store/{/DeepSpace9/Voyager/}/{/picture/video/sound/}/@
```



How compact and quick! We've declared two tag groups, one of two tags and another one of three, saving us from writing six (2 x 3) sub-queries joined by the +/ sign.

The only thing you can't do with tag groups is nest them. What would it mean anyway a tag group inside another one? Of course you are requested to close tag groups before completing the query. The result of ~/myfiles/store/{/startrek/@/ is left undefined.

Tagging is a day by day process. Today you tag some movie with a tag, tomorrow I'll want to add another one. How can be done this in Tagsistant?

There are two ways to add more tags to an object. The first one is to copy the same object in the new tag:

```
$ cp somefile.txt ~/myfiles/store/tag1/@
[ ... some days later ... ]
$ cp somefile.txt ~/myfiles/store/tag2/@
```

Tagsistant is smart enough to understand that you've copied the same file twice (see Deduplication later) so it keeps just one copy of the file with both tags tag1/ and tag2/.

This is however suboptimal, because you're forced to wait for the copy process to complete twice, very annoying especially for big files like movies. Moreover, deduplicating the files takes time too, because it has to scan the whole file to compute it checksum. A better way would be the mv command:

```
$ cp somefile.txt ~/myfiles/store/tag1/@
[ ... some days later ... ]
$ mv ~/myfiles/store/tag1/@/somfile.txt ~/myfiles/store/tag1/tag2/@
```

The mv command gets internally translated into a rename() call which Tagsistant manages very efficiently. rename() basically removes from somefile.txt all the tags listed in the source query (the left one) and then adds all the tags contained in the destination query (the right one). So what's basically happening is:

1. the tag tag1/ from the source query is removed
2. the tag tag1/ from the destination query is added
3. the tag tag2/ from the destination query is added

Point 1 and 2 produce a neutral effect, so the only result of this command is to add query tag2/ to somefile.txt. Just remember to add to the destination query the same tags you include in the source query, otherwise some tagging will be lost. To avoid unintentionally removing some tags, use the ALL/ special tag in the source query:

```
$ mv ~/myfiles/store/ALL/@/somefile.txt ~/myfiles/store/tag2/@
```

See later for more information on the ALL/ special tag.

Another frequent operation is merging two tags, because you find that one tag is a duplicate of another or any other reason. Mergin two tags is very simple with Tagsistant, it's just a matter of moving all the content of a tag (the merged) into another (the destination) and then delete the merged one:

```
$ mv store/merged_tag/@/* store/destination_tag/@/
$ rmdir tags/merged_tag
```

The use of @@/ in place of @/ may help in understanding and checking what's going on. @@/ disables the reasoner, as you'll read soon, and gives you a clearer view of what's tagged how in the end.

So far we have seen files being tagged by Tagsistant. But files are not the only object type Tagsistant can manage. In fact Tagsistant can tag any kind of object: directories, named pipes, devices and symbolic links. To create them you can use traditional Linux commands (mkdir, mkfifo, mknod, ln) with a complete query like in:

```
$ mkdir ~/myfiles/store/startrek/@@/subtitles/
$ ln -s ~/Videos/encounter_at_farpoint.avi ~/myfiles/store/startrek/video/@/
```

Here we have a directory named subtitles tagged as startrek/ and a link to a video tagged as startrek/ and video/ too. Symbolic links are very useful to tag a huge amount of files or big sized files in a matter of seconds. Let's focus on the video *encounter_at_farpoint.avi*. It's a big file (about 600 megabytes). Using the copy way you tag it as startrek/. Some days after you want to add the video/ tag, so you copy it again:

```
$ cp ~/myfiles/store/startrek/@/encounter_at_farpoint.avi ~/myfiles/store/video/@
```

Here you have to wait for all the 600 megabytes to be copied in the new directory and then deduplicated, which costs to Tagsistant the computation of the SHA1 checksum of the file (in other words a long and useless task). If however you would have used symlinks the operation would have taken the fraction of a second:

```
$ ln -s ~/myfiles/startrek/@/encounter_at_farpoint ~/myfiles/store/startrek/@
$ ln -s ~/myfiles/startrek/@/encounter_at_farpoint ~/myfiles/store/video/@
```

Tagsistant deduplicates symlinks by comparing their pointer object, which is an instantaneous operation. This approach has another advantage: you don't put your original files inside Tagsistant. Don't misunderstand me: Tagsistant is a safe place for your files. But if you put your files inside it then you have to use it to access them. This way you can keep accessing your files in the traditional way and in the tag way at the same time!

Let's do a small wrap up of the main concepts seen so far. Each directory under store/ or tags/ is a tag. If you copy a file under the store/ directory it gets tagged. You can establish relations between tags using the relations/ directory. If a tag A includes the tag B all the files tagged as B will show up in store/A/@/ as well.

So far, so good. Now imagine that you tagged your mp3 library by band name and then organized the band tags by genre. In the end you included all the genre tags in music. Something like:

```
$ cp the_number_of_the_beast.mp3 ~/myfiles/store/iron_maiden/@
[... other files too ...]
$ mkdir ~/myfiles/relations/heavy_metal/includes/iron_maiden/
[... other bands too ...]
$ mkdir ~/myfiles/relations/music/includes/heavy_metal/
$ mkdir ~/myfiles/relations/music/includes/jazz/
$ mkdir ~/myfiles/relations/music/includes/classical/
$ mkdir ~/myfiles/relations/music/includes/piano/
[... other genres too ...]
$ ls ~/myfiles/store/music/@
[... your whole library here ...]
```

Amazing! All your files in one place, without having to tag them as music one by one. Now you can for example open ~/myfiles/store/heavy_metal/@/ with your favorite audio player to listen to all your heavy metal collection, excluding those fine jazz sessions or Bach's fugues. You start the application, click on the_number_of_the_beast.mp3 and... the smooth timbre of a piano spreads in the room. What the hell happened to distorted guitars?

Oh, sure, now you remember: that version of The Number of the Beast is a nice tribute cover by a classical piano player. Better move it to ~/myfiles/store/classical/piano/@:

```
$ mv ~/myfiles/store/music/@/the_number_of_the_beast.mp3 ~/myfiles/store/classical/piano/@
```

You give the move command and... the file is still there?!?!

Of course it is, because the reasoner knows that the music tag includes both classical and piano tags too, so your file still features in the results of store/music/@. But how could you know that moving (re-tagging) the file happened? The reasoner prevents you from being sure.

The answer is: ask the reasoner to not step in!

If you end a query with the special @@/ marker, the reasoner doesn't get involved, so only the files with an explicit tagging are returned. In example, if you list store/music/@@/, no files are listed, because music is *by itself* totally empty (no file has been tagged as music). The very same happens if you list store/heavy_metal/@@/. But if you list store/iron_maiden/@@/, all your Iron Maiden songs are there.

The @@/ marker is usually applied to single tag queries, like store/iron_maiden/@@/ to ease the re-tagging process.

I'm sure you'll agree with me that Iron Maiden are a cult band that composed and performed immortal songs, but maybe you don't like all of their periods. Let's say you would like to exclude the albums from the Bayley's period. How could your do it?

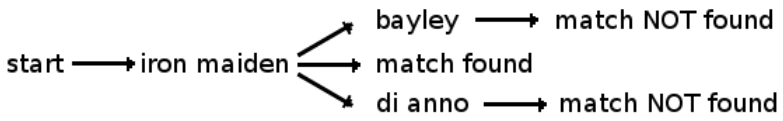
Provided that you tagged those songs with a "bayley" tag, you can use the -/ operator to exclude those songs from the results:

```
$ ls ~/myfiles/store/iron_maiden/-/bayley/@/
```

Think to the -/ operator like a filter. First all the songs tagged iron_maiden/ are selected and then all the songs tagged bayley/ are discarded from that selection.

The -/ operator applies only to the tag that follows it, so if you want to exclude another singer, you can do it with:

```
$ ls ~/myfiles/store/iron_maiden/-/bayley/-/dianno/@/
```



But pay attention: you have one more singer left. I've warned you.

When you write a query you can of course make some mistakes. For example you could start a query with the negation operator -/ or open a tag group inside another tag group. Tagsistant will refuse to process a malformed query and will inform you about the error. This happens by listing only one file as result, called error which contains a description of your error:

```
$ ls ~/myfiles/store/{/{/t1/@@
error
$ cat  ~/myfiles/store/{/{/t1/@@/error
Syntax error: nested tag group. Close all tag groups before opening another.
```

This works only in the store/ directory because it's the only one that's supposed to return a file set, but it's also the one where the biggest part of your errors will happen.

Now you perfectly know how to tag files and retrieve them by complex queries. What you may ask now is: how do I know what tags have been applied to a file?

This is somehow the complementary question to: what files are tagged as X and Y? The query syntax allows you to answer to this kind of questions, but does not answer to the first one. Hence Tagsistant provides you with a feature called tag suffix which is a string you can append to any object in a complete query to automagically create a file containing all the tags applied to that object. The default tags suffix is ".tags", but you can change it from the command line if you think this will conflict with your files. Let's see how it works:

```
$ cp ~/somefile.txt ~/myfiles/store/t1/t2/@@/
[ ... some time later ... ]
$ cp ~/somefile.txt ~/myfiles/store/t6/t7/@@/
[ ... some days later ... ]
$ cat ~/myfiles/store/t6/@@/somefile.txt.tags
t1
t2
t6
t7
document:type=text
```

This feature works for files, directories, symlinks and any other kind of object supported by Tagsistant. Tags suffixed files do not get listed in normal queries to avoid hogging up your results, but you know you can make them appear whenever you need them.

You may wonder how Tagsistant refers to the files it is managing. Well, whether you care or not, knowing how this happens is something very useful. Behind the scenes Tagsistant assigns to each file a unique number, called a Tagsistant inode. The word inode comes from the filesystem lingo and is used by Tagsistant in a rather liberal way. You can just think to it as an ID or a serial number, if this sounds more familiar. Just remember that each file has its own.

Now let's take a look inside the archive/ directory:

```
$ ls ~/myfiles/archive/
0 1 2 3 4 5 6 7 8 9
```

The archive/ directory is where your files are kept. Starting from release 0.8, Tagsistant organizes them in a hierarchy of subdirectories, placing each file in a specific directory chosen by reverting its inode and taking the first digits (by default the first three, but that can be configured). Let's see an example: a file with inode 1923 will be placed in archive/3/2/9/, because 3291 is the reverse of 1923 and only the first three digits are considered. In our examples we have a small amount of files and the situation is something like:

```
archive/1/1___first_contact.avi
archive/2/2___the_wrath_of_khan.avi
archive/3/3___the_empire_strikes_back.avi
```

When inodes are shorter than three digits, the hierarchy ends before. Here "first_contact.avi" has inode 1, "the_wrath_of_khan.avi" has inode 2 and so on.

You may be wondering why this is something you should know. In fact usually you don't care about inodes since you can see them just inside the archive/ directory, which you are not supposing to visit very often, being the big part of the game played inside store/ and tags/. This is correct, but with one important exception.

If two *different* files (this means two files with different contents), copied inside two or more different tags, have the same name (such as *avatar.jpg*), Tagsistant will apply the inode in front of them when it have to list both as the result of a search operation, like in:

```
$ cp ~/myblog/avatar.jpg ~/myfiles/store/blog/@
```

```
$ cp ~/movies/covers/avatar.jpg ~/myfiles/store/pictures/@
$ ls ~/myfiles/store/blog/+/pictures/@
... 231___avatar.jpg ... 862___avatar.jpg ...
```

This is the only way Tagsistant could let you distinguish two *different* files with the *same* name when both are returned by a query. If no filename overlaps, Tagsistant will avoid putting the inode:

```
$ ls ~/myfiles/store/blog/@
... avatar.jpg ...
$ ls ~/myfiles/store/pictures/@
... avatar.jpg ...
```

The two *avatar.jpg* files are not the same, but since there's no ambiguity, there's no need to show their inode.

After some time spent adding tags to your repository, you'll soon notice that the store/ and tags/ directories will get filled by a long flat list of tags which is difficult to manage. Simple tags are quick to write but will often *collide* (the same tag has different meanings in different contexts) or will loose their intuitiveness. How would you represent the year 2000? With the number 2000? As 'year_2000'? The second one is better, but implies the creation of a 'year_2001' tag and 'year_1999' tag too. Numbers are infinite, I don't need to explain why this is not the proper approach.

Tagsistant offers a second form of tags called machine tags, sometimes called also triple tags (because of their form) which better organize your tags in a more structured and clearer way. A machine tag is composed by three elements:

1. a namespace
2. a key
3. a value

The namespace describes the *semantic context* the tag belongs to. In the example of the year 2000, the context clearly is the measurement of time, so we could choose *time* as namespace. The choice of the namespace is up to you and you can use the one that better suites your taste.

The key describes which aspect of the semantic context the tag refers to. In this case the number 2000 is a year, so our key will undoubtedly be *year*.

The value is of course *2000* and I don't think I need to add anything else.

There is a fourth element which is a *comparison operator*, meaningful when the tag is included in a search query, which goes between the key and the value. This operator can be one from:

1. eq (equals to)
2. inc (includes)
3. gt (greater than)
4. lt (less than)

Knowing this, the complete syntax of a machine tag becomes:

```
namespace:/key/operator/value/
```

Please note the colon at the end of the namespace. This tells tagsistant that the first element starts a machine tag and must not be interpreted as a simple tag. For example a tag representing the year 2000 could be written as:

```
time:/year/eq/2000/
```

Now you know how to properly tag your last summer photos:

```
$ cp ~/summer_photos/*.jpeg ~/myfiles/store/time:/year/eq/2000/time:/month/eq/August/@@/
```

If you're thinking that tagging by date your photos this way is extremely boring, feel relieved: I agree. Tagsistant will do that for you with another feature: the autotagging plugins. You will not be forced to add yourself dates (or other numeric informations) to files that already contain them, like JPEG files do in their EXIF section. A dedicated plugin will scan your files looking for metadata in the very moment you copy your files under store/. More on this later.

Machine tags and simple tags can be mixed in any way you like:

```
$ mkdir -p ~/myfiles/store/time:/year/eq/1981/
$ mkdir -p ~/myfiles/store/iron_maiden/
$ mkdir -p ~/myfiles/store/music:/album/eq/killers/
$ cp ~/music/iron_maiden/killers/*.mp3 ~/myfiles/store/iron_maiden/time:/year/eq/1981/music:/album/eq/killers/
```

Here we tag (copy) all the MP3 files from Iron Maiden' Killers album as 'iron_maiden', 'music:/album/eq/killers/' and 'time:/year/eq/1981/' at the same time.

Now you may ask: what happens if I put twice the same file in two separate folders? Will Tagsistant create two copies of the same file or what? The answer is: as soon as Tagsistant notices that two files with the same content have been created, it deletes the second copy applying its tags to the first one. So if you do this:

```
$ mkdir ~/myfiles/store/movies/
$ mkdir ~/myfiles/store/startrek/
$ cp first_contact.avi ~/myfiles/store/movies/@/
$ cp first_contact.avi ~/myfiles/store/startrek/@/
```

right after the end of the second copy Tagsistant will compare the content of the two files *first_contact.avi*, guess the second is a duplicate of the first, delete the second and tag the first also as startrek/. So you can now:

```
$ ls ~/myfiles/store/startrek/movies/@/
first_contact.avi the_wrath_of_khan.avi
```

Deduplication can currently be a bit rough and confusing for the user. If two *identical* files named A.jpg and B.jpg get copied in two directories called tag1/ and tag2/, Tagsistant will delete B.jpg and tag A.jpg as tag2/ too.

```
$ cp A.jpg ~/myfiles/store/tag1/@/
$ cp B.jpg ~/myfiles/store/tag2/@/
[... deduplication happens here ...]
$ ls ~/myfiles/store/tag2/@/
... A.jpg ...
```

The *content* of B.jpg (being identical to A.jpg) is actually available under tag2/ too, *but as A.jpg*! The file ~/myfiles/store/tag2/@/B.jpg seems to have vanished. This is something Tagsistant will address in a future release.

Tagsistant features a stack of autotagging plugins based on libextractor. Thanks to its ability to extract metadata from a long list of file formats, Tagsistant is able to integrate the user tagging with some automatically provided information. Autotagging plugins are located in the src/plugins/ directory. Each plugin basically declares the mime-type it supports and sets a regular expression acting as a filter: if a key extracted by libextractor does not match it, that value is discarded and no tag is created. For example, a basic regular expression for the JPEG format could be "^(size|orientation)$" (which is actually the default one). The user can declare its preferred regular expressions in the repository.ini file, like in:

```
[mime:text/html]
filter=^(author|date|language)$

[mime:image/jpeg]
filter=^(size|orientation)$

[mime:application/ogg]
filter=^(year|album|artist)$
```

Version 0.5, 0.6 and 1.x of libextractor are supported. The list of plugins available so far includes:

1. application/xml
2. image/gif
3. text/html
4. image/jpeg
5. image/png
6. application/ogg
7. audio/mpeg

Information on writing plugin is provided here.

If you list the contents of the store/ directory you'll notice a tag named ALL/ you have not created:

```
$ ls ~/myfiles/store/
-  @  @@  +  ALL  startrek  starwars  scifi
```

The ALL/ tag is a special automatic tag which includes all the files managed by Tagsistant. When you list store/ALL/@@ or store/tag1/+/ALL/@ or any other combination you get the same results: all your files.

This tag may seem somehow useless: what's the need of a tag that lists all my files? You may think: I want to *narrow* the list of my files by tags, not list them all, otherwise I would not be using a tool like Tagsistant. OK, then answer to this question: how do you apply an action to all your files, knowing that a recursive scan of the store/ directory is impossible since it would travel all the infinite permutations of all your tags? The answer is: using the ALL/ tag.

```
$ grep "John Doe" ~/myfiles/store/ALL/@@/*.txt
```

A common operation is rescanning all your files for new tags when you add a new autotagging plugin:

```
$ touch ~/myfiles/store/ALL/@@/*
```

This will force a new scan and by the way redoes deduplication too.

The ALL/ tag is also useful to retag your files. For example to add the tag picture/ to a file named "IMG_09394.jpg" you first need to locate it somewhere. Let's say it's tagged holiday/, so:

```
$ ls ~/myfiles/store/holiday/@/
[ ... ]
IMG_09394.jpg # here you've found the file
[ ... ]
$ mv ~/myfiles/store/holiday/@/IMG_09394.jpg ~/myfiles/store/picture/@ # holiday/ tag disappeared!
```

You've moved "IMG_09394.jpg" inside picture/, but you've forgot to add holiday/ back in the destination query, mistakenly removing that tag from the file! A much better way is to access the file from the ALL/ tag and "move" it from there:

```
$ ls ~/myfiles/store/ALL/@/IMG_09394.jpg
IMG_09394.jpg
$ mv ~/myfiles/store/ALL/@/IMG_09394.jpg ~/myfiles/store/picture/@
```

The file got its new picture/ tag without the risk of removing any tag by mistake, since ALL/ is a virtual tag and can't be removed.

If you use a query very often, you may find annoying to enter it every time. Tagsistant offers the alias/ directory as a bookmark store to save your favorite queries. The alias/ directory contains files holding a query each. For example you could save an alias named *maiden_videos* like iron_maiden/file:/format/eq/AVI/ and another one named *summer_2000* like file:/format/eq/jpeg/time:/year/eq/2000/time:/month/eq/August/.

We should more properly talk of *fragments of queries*, because each alias can be combined with other aliases or extended by more tags when used inside a query. Let's see how.

If you list the content of the store/ directory, you'll see your aliases listed with regular tags but with an equal sign before them:

```
$ ls ~/myfiles/stores/
... =maiden_videos ... =summer_2000 ...
```

You can include your aliases in your regular queries:

```
$ ls ~/myfiles/store/=maiden_videos/@/
```

even combining them with regular tags:

```
$ ls ~/myfiles/store/=maiden_videos/time:/year/lt/1985/@/
```

The last one will be *expanded* before being processed as:

```
$ ls ~/myfiles/store/iron_maiden/file:/format/eq/AVI/time:/year/lt/1986/@/
```

(Long Beach Arena, we are coming!)

Ahem... Please use aliases with caution. They are actually replaced literally by their query fragment and this could hide some surprises, especially if the fragment contains the +/ operator. If the *maiden_videos* alias have contained:

```
iron_maiden/file:/format/eq/AVI/+/iron_maiden/file:/format/eq/MPEG
```

the query:

```
$ ls ~/myfiles/store/=maiden_videos/time:/year/lt/1986/@/
```

would have been translated into:

```
$ ls ~/myfiles/store/iron_maiden/file:/format/eq/AVI/+/iron_maiden/file:/format/eq/MPEG/time:/year/lt/1986/@/
```

The *time:/year/lt/1986/* tag would have been added to the second part of the query only! The Iron Maiden AVI files would have been returned even if filmed after the year 1986 while MPEG files would have been limited to that year!

Untagging a file or another object is as simple as deleting it. Don't worry: an object is actually deleted from Tagsistant only when it's removed from its last tag. As an example consider this situation:

```
$ cp /some/file.txt ~/myfiles/store/docs/texts/@/
$ rm ~/myfiles/store/docs/@/file.txt
```

Here file.txt has been untagged from docs but it's still recorded in the database and tagged as texts.

Deleting a tag is something very simple to do from the command line. A tag can be deleted using rmdir from the store/ or the tags/ directory:

```
$ rmdir ~/myfiles/store/useless_tag/
$ rmdir ~/myfiles/tags/another_useless_tag/
```

That's all!

However when using a filemanager you are forced to delete the tag from the tags/ directory. This is mainly the reason why the tags/ directory exists. Under store/ each tag contains all the other tags and operators to let you build your query path:

```
$ ls -l ~/myfiles/store/jazz/
total 1.7M
dr-xr-xr-x 1 tx0 tx0 140K Dec 19 00:14 -
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 @
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 @@
dr-xr-xr-x 1 tx0 tx0 140K Dec 19 00:14 +
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 =maiden_videos
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 iron_maiden
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 music
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 file:
drwxr-xr-x 4 tx0 tx0 140K Dec 19 00:14 time:
```

A file manager would probably refuse to delete a non empty directory or would ask the user to delete it RECURSIVELY! Of course this would WIPE OUT YOUR WHOLE REPOSITORY. Please pay a lot of attention when deleting something from the store/ directory and NEVER DELETE A store/ INCOMPLETE PATH. A path is incomplete when it does not include a @/ or @@/ mark.

To let you safely manage your tags without risking your entire repository, use the tags/ directory. Under the tags/ directory, tags have no contents. If you delete a tag from the tags/ directory you just delete that tag and nothing more, without affecting the rest of the repository.

The stats/ directory contains some special read-only files useful to get an idea of how Tagsistant is working. Let's see its content:

```
$ ls ~/myfiles/stats/
configuration  connections  objects  relations  tags
```

The configuration file contains the whole configuration Tagsistant is using, both compiled and runtime chosen:

```
$ cat ~/myfiles/stats/configuration

 --> Command line options:

        mountpoint: /home/tx0/myfiles
   repository path: /home/tx0/.tagsistant
  database options: mysql
 run in foreground: 0
   single threaded: 0
   mount read-only: 0
             debug: -
                    [ ] boot
                    [ ] cache
                    [ ] file tree (readdir)
                    [ ] FUSE operations (open, read, write, symlink, ...)
                    [ ] low level
```

```
                    [ ] plugin
                    [ ] query parsing
                    [ ] reasoning
                    [ ] SQL queries
                    [ ] deduplication

--> Compile flags:

 TAGSISTANT_ENABLE_QUERYTREE_CACHE: 0
    TAGSISTANT_ENABLE_TAG_ID_CACHE: 1
   TAGSISTANT_ENABLE_AND_SET_CACHE: 1
  TAGSISTANT_ENABLE_REASONER_CACHE: 1
TAGSISTANT_RETAG_INTERNAL_SYMLINKS: 0
        TAGSISTANT_VERBOSE_LOGGING: 0
        TAGSISTANT_QUERY_DELIMITER: @
       TAGSISTANT_ANDSET_DELIMITER: +
        TAGSISTANT_INODE_DELIMITER: '___'
```

The objects, tags and relations files contain the total number of entities in the database:

```
$ cat ~/myfiles/stats/objects
# of objects: 1744
$ cat ~/myfiles/stats/tags
# of tags: 46
$ cat ~/myfiles/stats/relations
# of relations: 39
```

Finally, the connections file contains a the total number of active database connections:

```
$ cat ~/myfiles/stats/connections
# of MySQL open connections: 1
```

You're now familiar enough with Tagsistant to know that a recursive scan of the store/ directory can't be done. This is not what indexing tools expect. updatedb (the companion of locate) and other tools like Baloo will try to descend the store/ tree, entering an endless loop. You should exclude every Tagsistant mountpoint from the set of scanned directories. For example, to exclude the Tagsistant managed /home/tx0/myfiles directory from updatedb, modify /etc/updatedb.conf and add the mountpoint to the PRUNEPATHSvariable like in:

```
PRUNEPATHS="/tmp /var/spool /media /home/.ecryptfs /home/tx0/myfiles"
```

To avoid scanning Tagsistant directories for each user, ask your users to mount Tagsistant filesystems in subdirectories of a directory called myfiles inside their homes and then add:

```
PRUNENAMES="myfiles"
```

Change the name of the mountpoint according to your tastes.

To unmount Tagsistant you can use the same command used for any other FUSE-based filesystem:

```
$ fusermount -u ~/myfiles
```

This command will kill the Tagsistant process and clear the mtab entry for you (if you don't understand, don't be scared, it's just stuff for the geeks).

While compiling Tagsistant you can choose to enable some experimental features by editing tagsistant.h in the src/ directory. The flags are those reported by the stats/configuration file in the compile flags section.

The only four flags you are supposed to tweak are:

1. TAGSISTANT_ENABLE_QUERYTREE_CACHE
2. TAGSISTANT_ENABLE_TAG_ID_CACHE
3. TAGSISTANT_ENABLE_AND_SET_CACHE
4. TAGSISTANT_ENABLE_REASONER_CACHE

Their purpose is to enable some caching layers to dramatically reduce the volume of SQL queries done. While the second and the third are stable and don't cause too much memory consumption, so being safely enabled on production, the first (the querytree cache) can cause Tagsistant to exhaust memory during huge data loading, so if you experience that problem, try disabling query tree cache.

The suggested configuration is:

```
  TAGSISTANT_ENABLE_QUERYTREE_CACHE: 1
     TAGSISTANT_ENABLE_TAG_ID_CACHE: 1
   TAGSISTANT_ENABLE_AND_SET_CACHE: 0
  TAGSISTANT_ENABLE_REASONER_CACHE: 0
```

Change it at your own risk.

In May 2013 I've done some tests on a particular situations: a chain of tag relations where t1 is included by t2 which is included by t3 ... which is included by tN, with N being 40. The objects managed were 8352! The test showed a very quick response time of 10.739s when Tagsistant was asked to:

```
ls ~/myfiles/store/t1/t2/t3/t4/t5/t6/t7/t8/t9/t10/t11/t12/t13/t14/t15/t16/
t17/t18/t19/t20/t21/t22/t23/t24/t25/t26/t27/t28/t29/t30/t31/t32/t33/
t34/t35/t36/t37/t38/t39/@
```

After the first run, issuing the same query again got answered in just 3.598s. This query if of course very suboptimal since gives the same results of ~/myfiles/store/t39/@, but my goal was to test how Tagsistant could behave under a lot of tags in the same query. The total files returned by the query were 937, which generated as much getattr (stat) calls to get result data (size, owner, permissions).

I've also successfully tested Tagsistant on repositories containing 100G of data.

I hope this quick introduction to Tagsistant 0.8 will be enough to let you experiment with the software and that you'll find Tagsistant useful.

If you have any comment, you're welcome on the Forum.