A light-weight non-hierarchical file system navigation extension

Jonas Amoson
Thomas Lundqvist

University West
Trollhättan, Sweden

{jonas.amoson | thomas.lundqvist}@hv.se

ABSTRACT

Drawbacks in organising and finding files in hierarchies have led researchers to explore non-hierarchical and search-based filesystems, where file identity and belonging is predicated by tagging files to categories. We have implemented a *chdir()* shell extension enabling navigation to a directory using a search expression. Our extension is light-weight and avoids modifying the file system to guarantee backwards compatibility for applications relying on normal hierarchical file namespaces.

1 Introduction

File systems have long been hierarchical, helping both system designers and ordinary users to group their files and thereby hopefully avoiding chaos. Besides this grouping of related files, a *file path* such as `/usr/$user/work/telephone.txt` is also important in that it uniquely identifies a file, directory or other named resource within a given name space, by a sequence of slash-separated strings [3]. While a well structured home directory serves to avoid clutter, it also means that the user will have to remember and type longer paths in order to specify files. This is because the same addressing mechanism is used both for storing files as well as for addressing them later on.

We believe that our user, in the everyday interaction with the computer, could afford some loss of addressing preciseness for the benefit of having to type less. Instead of entering an absolute or relative file path, the file or directory of interest may be specified using a much shorter *search expression*. For example, the command `cd !mydir` will then perform a search for a subdirectory matching `mydir` and make the found directory the new working directory of the process, even if it is located several subfolders deeper down from the current working directory. If the search matches multiple files, the user simply chooses the desired one, or refines the search.

The idea of regarding a path name as a search expression is old. Previous work in semantic file systems or non-hierarchical, tag-based file namespaces [1,2,4] has suggested many ideas and solutions for navigating through files and directories based on search expressions. However, these approaches are rather cumbersome to implement and fail to provide full backwards compatibility with hierarchical file systems.

This paper explores a more light-weight approach of interacting with an existing file system namespace, using the path name as a search string, but keeping backwards compatibility with the existing namespace. We argue that our search mechanism would be difficult to implement in the form of a separately mounted file system since the search capabilities we suggest are really a user interface feature. To explore our ideas, we have made a simple but yet useful prototype implementation in the form of a shell extension that enables powerful new ways of file system navigation.

2 The idea: a file system navigation extension

From a user's perspective, the search feature we propose is a simple extension of the syntax of path names used by the shell. In our implementation, we have picked the character "!" as a prefix to denote a search.

To illustrate the behaviour, consider the following set of directories:

```
/usr/jam/work/lectures/cs101/mydir/
/usr/jam/work/lectures/plan9/
/usr/jam/work/plan9/
```

Let us assume that the current working directory is `/usr/jam/` and that we issue the command `cd !mydir`. This will tell the shell to perform a search for a subdirectory with a name matching `mydir` and change the working directory to the specified directory. Further, the command `cd !cs101` would take the user to `/usr/jam/work/lectures/cs101` and not to its subfolder `cs101/mydir`, keeping the matching directory as shallow as possible.

If there is no unique directory match, the relative paths for all matching directories are listed, and the user could either use one of them, or refine the search by adding substrings separated by slashes to the search expression. The command `cd !plan9` would for instance match two directories, resulting in a failed command and a listing of possible alternatives. Refining the search to `cd !plan9/lect` would succeed and match the folder `work/lectures/plan9`. Only folders "below" the current working directory are considered in the search.

We believe that the possibility of searching directly in a path name is more convenient than the alternative of using existing tools for searching the file system. The command *du(1)* could be used by the user to find usable file paths, but the user would most likely not want to invoke such search commands in the midst of specifying the third file argument of a program to run. The handiness of fuzzy path searches can be compared with filename completion provided by `INS` in *rio(1)* or tab-completion in Unix shells.

A possible problem with our solution could be the use of directories or file names starting with the character "!", thereby needing some way of escaping this character. We are unsure about the magnitude of this problem however.

3 Related work

Since the invention of hierarchical file systems, many researches have pointed out various limitations in organising data storage files in a strict hierarchy. Already in 1991, Gifford et al. [2] presented the idea of a semantic file system where path names can be used as a search string by the user. For example, by writing `cd ext:/c`, you go to a virtual directory containing all files matching the search, in this case having the extension ".c". Their semantic file system idea represents a virtual read-only file system containing symbolic links that point to an underlying regular Unix file system.

Influenced partly by the collaborative *tagging* found in on-line community sites such as *flickr*[1], Stephan Bloehdorn and Max Völkel [1] implemented a file system *TagFS* where files are not organised in directories, but where each file is assigned multiple *tags* by the user. When a file is recalled later, the tags are used as parts of the path similar to an ordinary hierarchical file path. Walking the "directory" structure (`cd` *tag*) yields a search, and the contents (as displayed by `ls`) shows the possible tags that could be used to further narrow down the search. To achieve accessibility from different operating systems and easy integration over the Internet, *TagFS* is implemented as a WebDAV[2]-server using the http-protocol.

Margo Seltzer and Nicholas Murphy [4] also suggest the demise of the hierarchy in file storage, and give the ubiquitous Google-search as a prime example. Seltzer and Murphy have implemented a file system, *hFAD*, similar to *TagFS*, based on FUSE[3] under Linux.

4 Comparison with related work

Compared to these earlier approaches [1,2,4], our shell extension does not modify the underlying file system and thereby maintains full compatibility with an hierarchical namespace.

An important property of our idea is that the `cd` command will, as usual, take you to an existing directory. An incomplete search path will make the `cd` command fail. This means that we always have a well defined current working directory. The latter is important since some

[1] Flickr online photo management (http://www.flickr.com).
[2] Web-based Distributed Authoring and Versioning (http://www.webdav.org).
[3] Filesystem in Userspace (http://fuse.sourceforge.net).

filesystem operations, like creating new files, will be difficult to support otherwise. This can, for example, not be guaranteed for the semantic file system presented in [2] since a virtual folder can be a result of a search returning a union of files from multiple locations. This is true also for the two other approaches where a *directory* can be the result of a union of all files from multiple file system locations.

Another issue with previous approaches is that one file can be identified by many different path names, which could confuse an old-fashioned application that relies on unique file path names.

5 A modified shell implementation

As a first attempt of an implementation we have modified the *cd* (change directory) command in the shell *rc(1)* to accept a search expression prefixed by an exclamation mark.

In *rc*, the path is examined before calling *chdir(2)*. If the path is prefaced with an exclamation mark, the command *du(1)* is invoked to get a list of directories relative to the current working directory. The resulting directories that do not match all search criteria are discarded. Then, the list is further processed to prune sub directories that would overspecify the search. Finally, if only one path remains, *chdir()* is called with the remaining path, otherwise the candidate path names are printed to standard output. The source code for the shell extension and a helper program `dugrep` is available at `http://cumulus.ei.hv.se/~imjam/ref/spath`.

6 Discussion and future work

Our shell extension approach suffers from the nuisance that it is not always possible to narrow down the search, given a set of possible paths. For example, consider the two paths: `work/lectures/` and `lectures/work/`. Navigating using `cd !work/lectures` would match both paths without giving the user any possibility of further refining the search. One solution to the problem would be to force the user to resort to the normal *cd*-command and simply specify the precise path. Another solution would be to extend the search syntax and let the user choose among the alternatives in some way. An interesting observation here is that a set of well-organised directories would typically not trigger this problem.

There are promises in getting beyond the traditional file hierarchy, using a tagging filesystem, especially for certain types of personal files such as lecture notes and media files, but the non-hierarchical tagging ideas might not work as well for directories where the “belonging together” property of files is more important than the uniqueness of individual files, such as in a source code tree.

We believe that the search extension idea, as presented in this paper, would be difficult to implement as a synthetic file server. To do so, it would require some kind of “virtual folders” to represent searches, possibly leading to problems with file creation and unique identification of files.

In the meanwhile, we have already grown accustomed to the handiness and simplicity of the *search-path* extension, and will try to develop it further, with general file and directory search-expansion for all command-line arguments. Another open issue is how the our search capabilities can become integrated with filename completion as provided by `INS` in *rio(1)* or tab-completion.

References

[1] S. Bloehdorn, O. Gorlitz, S. Schenk, and M. Volkel. Tagfs – tag semantics for hierarchical file systems. In *Pro. The 6th International Conference on Knowledge Management (I–KNOW 06)*., 2006.

[2] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O’Toole, Jr. Semantic file systems. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, SOSP ’91, pages 16–25, New York, NY, USA, 1991. ACM.

[3] R. Pike and P. Weinberger. The hideous name. In *USENIX Summer 1985 Conference Proceedings*, page 563, Portland Oregon, June 1985.

[4] M. Seltzer and N. Murphy. Hierarchical file systems are dead. In *Proceedings of the 12th conference on Hot topics in operating systems*, HotOS’09, pages 1–1, Berkeley, CA, USA, 2009. USENIX Association.