Shifs

A language to build dynamic file servers

Uriel M.
uriel @ cat−v.org

ABSTRACT

Shifs allows you to easily transform and generate simple dynamic file systems that can be implemented directly from the command line and can take advantage of the standard tools used to manipulate text streams. *Shifs* is still only a prototype but it is already functional enough to be useful not only for experimentation but also in production environments.

1. Introduction

Shifs uses a very simple language to define a set of *patterns* that trigger *actions*. In that sense it is similar to *awk* but instead of handling text streams, it works with file trees. *Shifs* was born out of the desire to have a simple way to transform and extend the data exposed by file server in a generic and extensible way. Editing the source of existing file servers is too intrusive and makes it harder to maintain backward compatibility for applications that might expect the old interface. One could avoid that in some cases by adding new synthetic files to the file server that would export the data in the desired format, but that clutters the file server with files that might have a very specialized purpose that is not related to the function of the file server itself and would violate the core principle of doing one thing and doing it well. In other cases access to the file server might not be available, or one might want to transform a whole tree that might include files exported by different servers.

Another goal was to make it convenient to leverage the power of the standard application toolset of the operating system and in that way bring the *software−tools* philosophy to the world of dynamic file server creation, and do so in a secure and convenient way.

The more immediate task that prompted this interest was the need to overhaul the HTML generation code in *wikifs* which consists of almost a thousand lines of C code used to first parse the internal text format of wiki pages and then generate the HTML output. Considering that the same task could be accomplished by a dozen lines of awk and other standard tools, it appeared worthwhile to take advantage of that approach without discarding the convenience of a file system interface.

2. Suckfs

A first attempt was made to implement in C a complete language called *suckfs* (because it *sucks* files) in the spirit of awk where rules would match path names and actions would dynamically generate or transform the contents of the matched files.

But soon it became clear that the language would need to be too complex to perform even the most simple tasks, and it failed to accomplish the main goal that was to take advantage of existing tools and languages without adding extra complexity.

Also it was appreciated that it would be convenient if the facility could be available on as many platforms as possible. Inferno provided this and was a good opportunity to experiment with Limbo.

In hindsight this was a very good choice as it simplified some problems considerably. In particular using the *sh* module made everything much easier.

3. The shifs language

The language was simplified to the bare basics, providing a foundation that could be extended in the future as more complex interactions become feasible while delegating all the procedural heavy lifting to the Inferno *sh* module.

The language has a syntax somewhat similar to *awk* and *mk*; consisting of *rules*, each composed of a *pattern* and a set of *actions*.

3.1. Patterns

A pattern starts at the beginning of line and consists of an operation specifier (*r* for read, *w* for write, etc.), in the absence of any operation specifier *r* is used; followed by the regexp match operator ~ (other operators might be added in the future), the rest of the pattern up to the end of line or a trailing ~ is the argument to the regexp operator which is a regular expression that will be matched against file paths.

```
r ~[0-9]+/[a-z]+$
```

Figure 1. Sample shifs pattern

3.2. Actions

To complete a rule, a set of *action* lines follow the pattern line, action lines should begin with a *tab* character and use the Inferno sh syntax.

```
r ~^today$
	date | sed 's/ .*//'
```

Figure 2. A complete shifs rule

If the action is short it can also follow the second ~ in the pattern, for example the previous example could be rewritten as:

```
r ~^today$~ { date | sed 's/ .*//' }
```

Figure 3. A complete shifs rule in a single line

The braces around the action are handled by the sh module and are included for clarity.

The environment in which actions are run includes some convenient variables set by shifs before the action is run, of special interest is *$FPATH* which contains the full path that matched this rule.

4. Execution

It is easy to run *shifs* from a terminal window.

shifs [**−po**] [**−r** */source/tree*] [**−m** */mount/point*] [−f *rulesfile* | *rules*]

The *−p* option allows to 'pass−thru' files that don't match any rule so they are retrieved unchanged from the underlying file server. The *−o* makes the exported file system read only. The path passed to *−r* is used as starting point for the exported file tree, and actions use it as their working directory, if the *−p* options was used all the files under that tree that don't match any rule will also be accessible without any modifications, if *−r* is not used the current working directory is used instead. Rules can be specified directly from the command line or can be read from *rulesfile* if *−f* is provided, if neither is provided they will be read from stdin.

5. Walking, opening and reading.

Shifs keeps track of walk operations, when a *fid* is opened the full path is matched against each rule, if a rule matches a new process is spawned that sets its stdout to a pipe from where the subsequent read operations will feed and then executes the contents of the associated action.

6. Uses

Some of the tasks where shifs can be helpful are: performing simple transformations to the contents of arbitrary file trees, for example when importing a file system from a windows system one can easily transform all files to use the Plan 9/Unix new line format by simply using *tr(1)*.

Other useful task is to format data from its canonical format into formats convenient for export, a good example of this is adding html formatting before exporting a file tree over httpd.

Another possibility is to use it as replacement for *exportfs* that allows more fine grained control of not only what files are exported, but for example filter out part of the content by simply using *grep −v*.

One can easily log all file access to a file tree by exporting it thru shifs:

```
% shifs −o −r /lib/ '~
        { echo `{date} − $FPATH > /tmp/access.log; cat $FPATH }'
% cat /mnt/shifs/ndb/local > /dev/null
% cat /tmp/access.log
Wed Nov 15 14:58:52 CET 2006 − /ndb/local
```

Figure 4. A way to log access to a file tree

Also it can be convenient to create synthetic files that collect and summarize the contents of whole file trees.

Finally one can even create completely synthetic file trees, for example here is a file server that calculates the square of any given number:

```
% shifs '~^[0-9]+$~ {load expr; echo ${expr $FNAME $FNAME x}}'
% cat /mnt/shifs/4
16
%
```

Figure 5. A simple squarefs

7. Conclusions and future directions

The current version of *shifs* is still very much a prototype with many rough edges, but so far it has demonstrated to be a good tool to experiment with 'one off' file servers. And that is probably its greatest power, just like shells allow you to easily write small throw-away programs that when found useful can be codified either as shell scripts or in some cases rewritten in C. It also manages to at least partially succeed in its other goal of making the more traditional Unix/Plan 9 set of text manipulation tools more easily accessible when writing simple file servers.

Shifs is still somewhat fragile, but still can be useful in experimenting with changes to the interface of existing file servers and working as an adaptor between file system interfaces.

The main areas that still need work is handling of write operations and transformation of directory listings. The ruleset language is probably somewhat simplistic and naive and might need some redesign.

Perhaps for better integration with a Plan 9 environment a rewrite in C could be considered, but maybe it would be best to instead try to take advantage of the "native" Dis.

8. Acknowledgements

Awk is clearly the main source of inspiration, but the more general tool philosophy underlines the goals of the project. Also one must acknowledge how both Limbo and the sh module in Inferno made implementation almost trivial.

I am not aware at the moment of any similar projects, but it is very likely that someone has thought of a similar thing before, my apologies if I have not noticed it.

9. References

[Inferno] Inferno Manual page for Sh(1)

[Aho79] A. V. Aho and B. W. Kernighan and P. J. Weinberger ''AWK - A Pattern Scanning and Processing Language'',

[Kernighan76] B. W. Kernighan and P. J. Plauger ''Software Tools''