Abstract

As the deployment of servers and applications becomes more transient, the practices of system administrators have needed to adapt to be more agile. Most system administrators no longer edit the majority of configuration files by hand with a text editor, they use automation and configuration management tools like puppet, saltstack, ansible, and the like. Many utilities and daemons in the FreeBSD base system use their own custom configuration file format. While these various different formats are usually accompanied by man pages, they do not lend themselves to automation or programmatic editing. Space and tab delimited files make it harder to extract a specific value, and difficult to edit that value in place, whereas nested key-value pairs are easier to read, and are easily addressed using libUCLs dotted notation. To solve this, I propose teaching the various utilities and daemons in the FreeBSD base system to speak UCL – the Universal Config Language, as implemented by libucl. In addition, I propose adding two small tools to the base system to make the administration of such config files easier for humans and automated scripts.

1. Introduction

UCL (Universal Config Language) is an effort to define a modern configuration syntax and implement a library to parse it, that can be reused by many different applications to simplify administration. Inspired by the NGINX and bind syntax, with elements borrowed from JSON, UCL strives to strike a balance between human writability, machine readability, and compatibility with existing formats. libUCL can read UCL, JSON, and YAML, parse them into objects that can be read or manipulated, then emit the resulting objects back out in any of the three formats.

2. Advantages of UCL

There are a number of reasons to use UCL over existing formats like JSON. The biggest drawback to JSON as a configuration language is the lack of support for inline comments. UCL allows both single line (#) and multi-line (/* --- */) comments, has a more forgiving syntax and includes useful syntax sugar including:

- Optional quoting (foo = bar; is valid, unlike in JSON)
- Optional comma separation (items in an array do not need to be separated by commas)
- Optional trailing comma (after the last element in an array, which is not allowed in JSON, increasing the 'diff' when additional items are added to the array)
- Can use key = value; or key: value;

- useful suffixes: k (*1000), kb (*1024), s (second), min (minute), d (day), etc
- Multiple Boolean expressions: yes/no, true/false, or on/off
- It is still possible to treat numbers and booleans as strings by enclosing them in double quotes.
- Ability to include files (optionally files matching a glob pattern, or remote files with secure signature checking)

[Figure 1 – An example UCL config file]

3. Considerations for Adopting UCL

Adopting UCL for various utilities will allow much more flexibility in the configuration. Traditionally, utilities like newsyslog read from a single configuration file (`/etc/newsyslog.conf`). This file is shipped as part of the base system pre-populated with a number of defaults, rotating the logs generated by the basesystem. Recently, newsyslog was extended to also read additional config files from `/etc/newsyslog.d/` and `/usr/local/etc/newsyslog.d/`. This allows newly installed ports, like NGINX, to automatically deploy an additional fragment of configuration that will rotate the log files that will be created. This helps a great deal, but still poses a problem when the administrator wants to change a value in the default configuration file. Care will have to be taken when the system is next upgraded to ensure the change is not lost. libUCL provides two important features that allow us to better solve this problem. The first is a more flexible include system, and the second is a 'priorities' system. When conflicting keys are defined in an included file, the priorities system decides which key wins. If the newly included file has a higher priority, it is merged with the existing object, overwriting any conflicting values. If newsyslog were converted to UCL, the default newsyslog.conf that ships with the base system could be moved to /etc/default, and contain those default entries. Then /etc/newsyslog.conf (plus /etc/newsyslog.conf.d/ and /usr/local/etc/newsyslog.conf.d/) with sequentially higher priorities. Now if the administrator wants to override the default number of previous /var/log/messages files that are retained they can add an entry to one of those configuration files that defines only the 'count' parameter. The resulting configuration will be the keys inherited from the default and the count overridden by the higher priority config file. Each entry will also include an 'enabled' flag, which can be switched off in a subsequent configuration file, effectively deleting the default entry, but without having to resort to editing the default file that

ships with the base system. [Figure 2 and 3 – examples of the newsyslog.conf converted to UCL, and a fragment overriding one of the default configuration options]

This introduces a new challenge, because the configuration of newsyslog can now be scattered across multiple files, and multiple directories full of additional files, administrators need a tool to view the 'effective' configuration. Something along the lines of Samba's testparm, which parses the defaults and each of the configuration fragments, and outputs the final configuration as it will be viewed by the utility that is actually using the resulting configuration. In addition, this tool can apply libUCLs schema validation rules, allowing it to check the config file for general syntax errors, but also for the structure and validity imposed by the utility that will consume the config file. This can be used by the startup script to validate the configuration before stopping a running daemon when a restart or reload has been requested.

4. Using UCL

The other tool required to actually make UCL a useful configuration language is the ability to easily parse, extract fragments and values, and to change them programmatically. To this end, I have developed **uclcmd**, a command line interface to libUCL. In addition to making is easy for shell scripts to parse UCL, extract individual keys, loop over arrays, etc., it also allows for the scripted modification of a UCL config file.

[Figure 4 – example UCL config file, extracting a value, and looping over an array] [Figure 5 – modifying a UCL config file by merging an object containing only a subset of the keys]

5. The Transition

Changing the configuration format of the utilities in the base system will be a large undertaking, and introduce obvious issues with upgrading systems in place. To mitigate this, all utilities will retain their ability to parse their original config format. Only when a config file contains a UCL sentinel on the first line, something like: "#freebsducl1.0" will the file be parsed as UCL. This allow allows for the structure of the file to be changed in the future with less hassle, while maintaining the backwards compatibility.

6. Additional Considerations

Many utilities are growing support for libxo, which allows them to output data

in various formats including JSON. Since libUCL can both input and output JSON, the command line tools provided by this project will have numerous other uses as well. [Figure 6 – wc output in JSON, extract a specific value with **uclcmd**]

7. Target

There are a number of tools high on my priority list for conversion, these include:

- newsyslog
- crontab
- iscsi / ctld
- autofs
- freebsd-update
- portsnap

The other goal of the project is to convince more 3rd party applications to switch to libUCL, to make the ecosystem and the tools more widely adopted and therefore more useful.

Additionally, I am working on a puppet module that uses the 'resultant configuration' tool to compare the current configuration to that desired by the puppet manifest, and then uses **uclcmd** to apply the required changes to a config fragment. Similar example implementations for other automation tools are also on the road map.

8. Implementation

The first utility converted to using libUCL was newsyslog. The original configuration syntax left much to be desired. A mix of spaces and tabs delimit the fields, there are too many fields, so the content wraps on a standard 80x24 terminal, and the second to last optional field can have two entirely different meanings. The flags field, is made up of a series of individual letters, that are hard for a user to understand at a glance.

[Figure 7 – Old newsyslog.conf]

The new syntax is much easier to read, less repetitive, and can be understood without knowing the meanings of magic letters. The flags field is replaced by a series of boolean values such as: `binary = yes;` and `create = yes;`, and `compress = xz;` or `compress = gzip;` innately eliminates the possibility of conflicting flags. The `path_to_pid_cmd_file` field is replaced with two separate keys, and `signal_number` can take an integer or a symbolic name.

[Figure 8 – new newsyslog.conf]

The implementation in newsyslog is quite simple, in the function `parse_file()` before the config file is read, I read the first line of the file, and check it for the sentinel "#freebsducl1.0". If the sentinel is

found, parsing of the config file is passed off to a new `parse_ucl()` function, otherwise, `rewind()` is called on the file, and the normal parsing routine continues as before.

[Figure 9 – which config format is this file?]

Then it is just a matter of loading the internal struct with the correct values, applying validation where needed, and doing some translation for things like the signal number.

[Figure 10 – some example code for doing validation]

9. Future Work

- Finish converting more utilities
- Fine the config format and lock it down for 11.0-RELEASE
- Spread the word and expand adoption of libUCL
- Implement **uclcmd** in more config management frameworks