Practical Perl Tools

Mind Those Stables, I Mean Files

DAVID BLANK-EDELMAN

David N. Blank-Edelman is the director of technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs. David is honored to have been the recipient of the 2009 SAGE Outstanding Achievement Award and to serve on the USENIX Board of Directors beginning in June of 2010.
dnb@ccs.neu.edu

In the last column, we had a lovely visit with the XML Path Language, or XPath, followed by a brief look at how to bring its power to bear using Perl. Path-like things have been a bit of a leitmotif here over the past few columns, so I thought we might want to continue in this direction. This time we're going to look at a library/tool that takes this abstraction and puts it to practical use in an interesting way we haven't seen yet. As regular readers of this column will tell you, I loves me a good abstraction now and then. Sure do.

Just a quick warning: like last column, the majority of the words will describe the tool/API because once you have that all down pat, bringing Perl into the picture is both easy and a bit of an anti-climax (until you realized how cool what you just did with a single line of code actually is).

Augeas

The tool I have in mind to explore today is called Augeas. Augeas was developed under the aegis (oh, it is just raining Greek mythology references today) of the Red Hat's Emerging Technologies group who have sponsored a number of spiffy projects. Determining exactly what Augeas is can be tricky because the project site [1] describes the tool as:

- An API provided by a C library
- A command line tool to manipulate configuration from the shell (and shell scripts)
- Language bindings to do the same from your favorite scripting language
- Canonical tree representations of common configuration files
- A domain-specific language to describe configuration file formats

Or more concisely, "Augeas is a configuration editing tool. It parses configuration files in their native formats and transforms them into a tree. Configuration changes are made by manipulating this tree and saving it back into native config files."

But that still probably doesn't make things crystal clear unless you've already had the itch this is designed to scratch. I think I can demonstrate the problem to you so you can see exactly why this particular solution is so lovely. Let's take a little tour together around the /etc directory of your average UNIX root filesystem (kindly keep your hands inside the car at all times and don't forget to stop at the gift shop on the way out).

First stop, an excerpt from the local hostname to IP address mapping file, /etc/hosts::

```
127.0.0.1 localhost
127.0.1.1 precise32
```

Next some lines from the file that contains the authentication information for all of the local users on a machine, /etc/passwd. Here's an excerpt from the stock Ubuntu 12.04 password file::

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

Here are some lines from the config file for the service that runs commands at a set interval or time:

```
    # m h dom mon dow user        command
    17 *   * * *  root    cd / && run-parts --report /etc/cron.hourly
    25 6   * * *  root   test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.daily )
```

And finally, a few lines from the config file for the service that rotates log files:

```
/var/log/wtmp {
    missingok
    monthly
    create 0664 root utmp
    rotate 1
 }

 /var/log/btmp {
    missingok
    monthly
    create 0660 root utmp
    rotate 1
 }
```

As much as it might be fun to try and write an entire column just by quoting files, I think this is probably enough to make my point. You don't have to be a seasoned sysadmin to see that none of these file formats look the same. And although a seasoned sysadmin can read, understand, and write these different formats in her or his head, getting a single piece of code to do that is much trickier. I've written my share of individual pieces of sed/awk/cut/Perl code to parse and rewrite all of these formats and more. Many times I've had to start from scratch because the formats were more different than similar.

Here's where Augeas comes into the picture. Augeas provides a framework for creating plugins (which they call *lenses*, a term I don't like all that much) that can read and write the formats for all of these different kinds of files and many more. When Augeas reads in the data from a file, it transforms the data into a tree that you can manipulate. When Augeas writes that tree data back out again, the data is written in the native format for that file.

Tree Talk

Just like we had to do some head scratching in my last column about how to go from a XML or HTML document to a tree, again I think in behooves us to make sure we get the concept. In fact, extra scrutiny is warranted in this case because that tree's construction and manipulation is central to the whole process.

Augeas trees always have at least two children at the top level: /augeas and /files. /augeas is a place to find information about augeas operations and configuration. Information like which lens is being used to parse a file, any errors reported in that parsing, global configuration information, etc. all live in this part of the tree. We're not going to talk about this branch at all in this column, but knowing about it may come in handy for you some day.

The branch of the tree that interests us the most is /files, where you can find (you guessed it) the configuration data found in your files. If we wanted to see the data from the files above, we would look in:

```
/files/etc/hosts/
/files/etc/passwd/
/files/etc/crontab/
/files/etc/logrotate.conf/
```

respectively. This list of paths brings up two interesting points:

1. We'll be referencing our configuration data using its position in the filesystem. Why is that so interesting? It is worth noting because it give you a sense of what Augeas is *not*. Augeas is not trying to be an all-encompassing abstraction framework that elides all of the details of how a system is implemented. If you are using a system that keeps its hosts file in /var/etc/hosts for some bizarre reason, Augeas will not "standardize" the data by putting it in /files/etc/hosts. At best Augeas can be taught to use the "hosts" lens when it sees something in /var/etc with that name, but it isn't going to provide a standardized tree independent of the underlying filesystem details.
2. I left the trailing slash on those paths to pique your curiosity. All of the data for each file lives in a sub-tree that starts with the name of the file. From that point in the tree, we see a further elaboration of point #1. The data found in the different file branches is represented in a way best suited for each file type. Again, pay attention to what Augeas is *not* doing here; it is not trying to represent every configuration file format in some uniform tree structure.

To see best what I mean in detail #2, let's look at the sub-trees from the first two files in our list. I'm going to reproduce just the part of the tree related to the lines I excerpted above. In /files/etc/hosts, we can see:

```
/files/etc/hosts
/files/etc/hosts/1
/files/etc/hosts/1/ipaddr = "127.0.0.1"
/files/etc/hosts/1/canonical = "localhost"
/files/etc/hosts/2
/files/etc/hosts/2/ipaddr = "127.0.1.1"
/files/etc/hosts/2/canonical = "precise32"
```

In /files/etc/passwd, we see this:

```
/files/etc/passwd
/files/etc/passwd/root
/files/etc/passwd/root/password = "x"
/files/etc/passwd/root/uid = "0"
/files/etc/passwd/root/gid = "0"
/files/etc/passwd/root/name = "root"
/files/etc/passwd/root/home = "/root"
/files/etc/passwd/root/shell = "/bin/bash"
/files/etc/passwd/daemon
/files/etc/passwd/daemon/password = "x"
/files/etc/passwd/daemon/uid = "1"
/files/etc/passwd/daemon/gid = "1"
/files/etc/passwd/daemon/name = "daemon"
/files/etc/passwd/daemon/home = "/usr/sbin"
/files/etc/passwd/daemon/shell = "/bin/sh"
```

Let me draw your attention to one similarity and one difference between the two trees. Both file trees have leaves representing the different fields of their respective records. For /etc/hosts, you can see a leaf for each IP address; for /etc/passwd, you can see a leaf for each uid, gid, shell, etc.

But now a key difference to note: the first file has a sub-tree for every line in the file, and the second has a sub-tree for every login name. In the first case, the lens author has chosen to let you distinguish the "record" you are working with by line number (.../1/..., .../2/..., and so on); in the second case, you would specify the record of interest using the login name instead of its place in the file.

Eagle-eyed readers are no doubt thinking, "Hey, everything has been bunnies hopping through the meadow so far, but what happens when the records are duplicated or a record has multiple leaves of the same type?" Ok, let's find out. Let's edit our hosts file to have this as the first line:

```
127.0.0.1       localhost huey dewey louie
```

and let's change /etc/passwd to have two bin accounts (not a good idea, I know):

```
bin:x:2:2:bin:/bin:/bin/sh
bin:x:22:2:bin:/bin:/bin/sh
```

Augeas borrows the numeric predicate notation from the XPath playbook to handle these cases and creates sub-trees like this:

```
/files/etc/hosts/1
/files/etc/hosts/1/ipaddr = "127.0.0.1"
/files/etc/hosts/1/canonical = "localhost"
/files/etc/hosts/1/alias[1] = "huey"
/files/etc/hosts/1/alias[2] = "dewey"
/files/etc/hosts/1/alias[3] = "louie"
```

and:

```
/files/etc/passwd/bin[1]
/files/etc/passwd/bin[1]/password = "x"
/files/etc/passwd/bin[1]/uid = "2"
/files/etc/passwd/bin[1]/gid = "2"
/files/etc/passwd/bin[1]/name = "bin"
/files/etc/passwd/bin[1]/home = "/bin"
/files/etc/passwd/bin[1]/shell = "/bin/sh"
/files/etc/passwd/bin[2]
/files/etc/passwd/bin[2]/password = "x"
/files/etc/passwd/bin[2]/uid = "22"
/files/etc/passwd/bin[2]/gid = "2"
/files/etc/passwd/bin[2]/name = "bin"
/files/etc/passwd/bin[2]/home = "/bin"
/files/etc/passwd/bin[2]/shell = "/bin/sh"
```

This means we will use a number within a square bracket to distinguish which branch of the tree we care about. An important thing to note here is Augeas is strict about ordering within the tree. Lines from a file go into the tree—and come out of it—in the same order they are found in the file.

Augeas' Muse, XPath

So you don't think the XPath reference a moment ago is unintentional, Augeas lets you reference things such as

```
/files/etc/passwd/bin[last()]
```

to get the last *bin* line from the configuration data. And indeed, we can now bring some of the power of XPath's query capability we saw in the last column to Augeas. For example, we could request all of the logins with the bogus shell of /bin/false:

```
/files/etc/passwd/*[shell = '/bin/false']
```

One example from the Augeas documentation (slightly modified):

```
/files/etc/hosts/*/ipaddr[../alias='huey']
```

This query finds the IP addresses of the host with the alias *huey*.

There are a bunch more selection and querying operations you can do using a syntax that is a kissing-cousin to XPath. Rather than enumerate them all, I'd like to point you to last *;login* issue's Practical Perl Tools column and the Augeas documentation. Now let's look at what you can do with everything we've discussed so far. For instance, what if you could do something you couldn't do with XPath, such as change values and whole records?

Let's Do It

Changing things sounds pretty spiffy, no? Let's get at it. The way I recommend you start working with Augeas is to install it on a machine using either a pre-built package or grab the latest source from github [2]. For build instructions, see the HACKING [3] file once you clone the repository.

Two quick OS X-related warnings as of this writing:

1. A key part of Augeas 0.10.0, as made available through both Homebrew and MacPorts, builds fine but fails as soon as you try to run it. If you want to run Augeas on OS X, you'll have to build from source, which leads to:
2. The latest version of Augeas requires a more recent version of Bison than the OS X default of 2.3. MacPorts can build a compatible Bison version; Homebrew mainline does not have that available.

For more details on where to get Augeas, including the language-specific binding (Perl is only one of many available), see their download page [4]. You may also be able to download the language binding of your choice from the same pre-built source as your main Augeas distribution. For example, Ubuntu makes the libconfig-augeas-perl package available.

The main distribution comes with a nifty command-line tool called augtool, which is a great way to play around with Augeas in an interactive setting. One hint if you are going to do this: augtool can take a -- root flag that will chroot() it to a directory of your choice. This lets you play with augtool on a directory of configuration files mocked up to look like a real root filesystem (one such directory comes with the distribution) without worrying about zorching your real configuration data. Another good way to experiment is within a throw-away virtual machine.

Augtool has built-in help, but mostly you can stick to a few simple commands:

- *ls* to show an Augeas path (e.g., ls /files/etc/passwd/bin)
- *print* to recursively print out the contents of a sub-tree, which is the command I used for the previous output
- *match* to do an XPath-like search on the tree
- *set* to set a value (more on this later)
- *rm* to remove an entire sub-tree (e.g., to remove a line from a config file)
- *ins* to insert a sub-tree (e.g., to insert a line into a file)
- *save* to write the tree data back out to a file with all changes made

Let's play with a few of these commands. I promised change, so let's change one of the alias values in the first line of the hosts file we were using:

```
$ sudo augtool
augtool> set /files/etc/hosts/1/alias[3] 'phooey'
augtool> save
Saved 1 file(s)
augtool> quit
$ grep phooey /etc/hosts
127.0.0.1 localhost huey dewey phooey
```

Here we've set the third alias value for the first line and confirmed that change actually took. Now, let's deal with that icky duplicate bin entry in /etc/passwd:

```
$ sudo augtool
augtool> rm /files/etc/passwd/bin[2]
rm : /files/etc/passwd/bin[2] 7
augtool> save
Saved 1 file(s)
augtool> quit
$ grep bin:x /etc/passwd
bin:x:2:2:bin:/bin:/bin/sh
```

Gone, baby gone. Augtool also has told us the size of the sub-tree we just removed.

Augtool also works for searches like the ones I mentioned above:

```
augtool> match /files/etc/passwd/*[shell = '/bin/false']
/files/etc/passwd/syslog = (none)
/files/etc/passwd/messagebus = (none)
/files/etc/passwd/ntp = (none)
/files/etc/passwd/vboxadd = (none)
/files/etc/passwd/statd = (none)
```

The = `(none)` part of the results is meant to indicate that the nodes selected by the query do not have values of their own (i.e., they are at the top of their respective file tree). Here's an example where we are asking for something that does contain a value:

```
augtool> match /files/etc/hosts/*/ipaddr[../alias='huey']
/files/etc/hosts/1/ipaddr = 127.0.0.1
```

Using This from Perl Is Going to Be Trivial, Right?

Yeah, yeah it is. All of the things we were just doing in augtool look remarkably similar using the Config::Augeas Perl module:

```
use Config::Augeas;

my $aug = Config::Augeas->new();
$aug->set('/files/etc/hosts/1/alias[3]', 'kablooey');
$aug->save;
```

Or, another example

```
use Config::Augeas;

my $aug = Config::Augeas->new();
my @matches = $aug->match(q{/files/etc/passwd/*[shell = '/bin/false']});
print join ("\n",@matches);
```

yields

```
/files/etc/passwd/syslog
/files/etc/passwd/messagebus
/files/etc/passwd/ntp
/files/etc/passwd/vboxadd
```

Don't you love it when a plan comes together? Now you have super powers when it comes to reading and editing system config files from Perl thanks to Augeas. Take care and I'll see you next time.

References

[1] Augeas: http://augeas.net/.

[2] Augeas on github: https://github.com/lutter/augeas.

[3] HACKING: https://github.com/lutter/augeas/blob/master/HACKING.

[4] Augeas download page: http://augeas.net/download.html