Tip: Command-line XML processing

Working with XML documents from the shell

David Mertz

May 07, 2003

Most of the time, processing XML documents utilizes heavy-duty APIs and custom applications. However, the tradition of using small tools with I/O piped between them works fine on Unix-like platforms. Here, David shows you how you can use XML for this kind of quick-and-dirty processing with one-liners that are especially useful during development and debugging cycles.

As much as I hate to say it, XML tools simply have not reached the level of convenience of text utilities that are available at a Unix-like command-line. For line-oriented, whitespace- or comma-delimited-text files, it is quite amazing what you can accomplish with clever combinations of `sed`, `grep`, `xargs`, `wc`, `cut`, pipes, and short shell scripts.

In my opinion, it is not that XML is inherently resistant to the modular treatment flat text files enjoy; XML developers just need to learn from experience the best ways to componentize XML tools. For example, in writing this tip I had a few realistic sample tasks in mind; but what I found was that even those tools that have command-line facilities have not yet learned to play nicely with each other. Working with multiple tools is not intractable, it just requires a little bit of wrapping.

One fact worth noting is that quite a few people have written versions -- in various programming languages -- of similar simple tools. Each version behaves a bit differently, but they all tend to accomplish the same overall task. For this tip, I look at the tools `xml_indexer`, `xmlcat`, and `xpath`; the first two come from my Gnosis Utilities, while the last is a Perl module written by Matt Sergeant (get it from CPAN).

Finding words in XML prose

I have written previously about `xml_indexer` (see Related topics), which creates an index of the words within XML documents by their XPath. For example, you can index then search an XML document with:
Tip: Command-line XML processing

Listing 1. Indexing and searching on XPaths

```
% xml_indexer chap.xml
% indexer events were
1 files matched wordlist: ['events', 'mostly']
Processed in 0.062 seconds (SlicedZPickleIndexer)
```

These commands display the elements within the XML document chap.xml that contain the words "events" and "were" (not necessarily in order or proximity). If other XML documents were added to the index, matching occurrences in them would also appear. By the way, new searches are almost instantaneous, even if multiple documents are indexed.

While it tells you a little bit to know that words occur at particular XPaths within particular documents, the point of a search is usually to see (or further process) the actual content matches. For that, you need to employ a command-line `xpath` tool; I have installed Perl's `XML::XPath`, whose behavior I like.

You can cut-and-paste the discovered XPaths into the `xpath` tool. For example:

Listing 2. Manually looking at an XPath

```
% xpath chap.xml '/chapter/sect1[2]/sect2[4]/para[3]'
Found 1 nodes:
-- NODE --
<para>It is not particularly remarkable that...
...</para>
```

This points to a nice modularity in the tools. Moreover, if the XPath that's passed to `xpath` were to have wildcards in it, it might match more than just the one node. Unfortunately, the output of `indexer` does not have quite the right form to pipe to `xpath`; to automate looking at the nodes with matched words, `indexer` separates the filename from the XPath with "::", and `xpath` only looks at one XPath at a time. You can do better.

A first little shell script

You might find a way to manage the impedance mismatch above using clever combinations of `xargs`, `apply`, pipes, and the like. But I found it easier to write a short (and reusable) shell script:

Listing 3. find-xml-elements

```
#!/bin/sh
for hit in `indexer $@ 2> /dev/null`
do
    echo $hit | sed 's/::/ /' > loc.tmp
cat loc.tmp | xargs xpath 2> /dev/null
    echo
done
echo
echo
```

As with other well-designed command-line tools, `indexer` and `xpath` send informational messages to STDERR, and the actual results to STDOUT. For my script, I am not interested in the STDERR messages. Now I can find all the nodes in which a list of words occur as easily as:
Tip: Command-line XML processing

Listing 4. Searching XML elements for words

% find-xml-elements events were
<para>Lest we forget some events in a recent decade...
...
Salem and by HUAC.</para>
<para>It is not particularly remarkable that...
...
being uncovered.</para>

So far, so good. The search outputs a series of XML snippets, where each top-level element contains the searched words. However, the result is generally not quite a well-formed XML document, since it is multiply-rooted.

Comparing XML documents and extracting text

One of the challenges of analyzing XML data is that XML documents can contain variations in formatting that are irrelevant to their semantic content: Some whitespace can be ignored, the order of attributes is discarded during parsing, empty elements may be either self-closed or have an end-tag, and entities can be encoded in a few different ways. In truth, even much of the whitespace that can't be ignored from a parser's perspective is nonetheless insignificant from an application point-of-view; pretty newlines and indenting are useful for people, and many applications (optionally) perform such stylistic formatting.

A rather large number of tools have been written to compare XML documents in a semantically useful way. Most of them have been given an obvious name like xmldiff (use Google to find versions for various programming languages). Underlying such a comparison of XML documents is a canonicalization of the layout of each document. Once inflexible algorithmic decisions have been made about the exact rendering of an XML document, semantically similar documents are easier to compare with generic tools like diff.

I use a Python script I wrote called xmlcat. The tool is not complicated -- it acts much like the standard cat utility -- but canonicalizes XML documents along the way. The reason I like xmlcat more than similar tools like xmlpp (see Related topics) is that it adds an option that's inspired by the Web browser lynx. If you pass the --dump argument to xmlcat, it outputs only the textual content of an XML document, eliminating the tags (using vertical whitespace is a moderately pretty way to do this). For data-oriented XML this capability is of little use, but for marked-up prose it is handy.

A second shell script for viewing text

If you search XML documents of prose for content words, most likely you are interested in the content more than you are the markup. Filtering with xmlcat --dump is exactly the trick to remove unwanted XML tags. However, directly piping the output of find-xml-elements to xmlcat is not quite right, since the output of find-xml-elements is not quite an entire well-formed XML document (it is fragments, as noted). A short shell script solves the problem:
Listing 5. find-xml-text

```bash
#!/bin/sh
for hit in `indexer $@ 2> /dev/null`
do
  echo $hit | sed 's/::/ /' > loc.tmp
  cat loc.tmp | xargs xpath 2> /dev/null | xmlcat --dump
  echo
done
rm loc.tmp
```

The output from `find-xml-text` plays nicely with standard text utilities. For example, I would like to display all the paragraphs that contain some search terms, but remove any left indent from their lines and limit line-length:

Listing 6. Searching XML element text for words

```
% find-xml-text events were | sed 's/^ *///' | fmt -w 70
Lest we forget some events in a recent decade...
  ...
  ...those in Salem and by HUAC.
It is not particularly remarkable...
  ...
  ...being uncovered.
```
Related topics

• Find the Perl tools, xmldiff (compare XML documents) and xmlpp (XML pretty printer), at the DecisionSoft site.
• IBM trial software: Build your next development project with trial software available for download directly from developerWorks.