XMLStarlet: a Unix toolkit for XML

An introduction to a quick solution tool that allows manipulating verbose XML files with a minimum of typing

Alexandre Rafalovitch

XML is everywhere. A quick Google search shows more than a *100 Million* articles about the subject. The XML proponents gush about its ability to provide structure and yet remain human readable. The XML critics are quick to mention that XML is so verbose that being human readable does not necessarily make it human comprehensible. Both sides are correct. Yet, despite the ongoing arguments, XML is already integrated into many software products and the rate of adoption is still on the rise. And that means that you need to learn tools and techniques that will allow you to use XML effectively.

XML is already integrated into many software products and the rate of adoption is still on the rise. And that means that you need to learn tools and techniques that will allow you to use XML effectively

There are many ways to work with XML. If you need an overview of tools and techniques, Free Software Magazine published one in its first issue (XML: The answer to everything? (http://www.freesoftwaremagazine.com/free_issues/issue_01/focus_format_xml/)). In this article I will be using XMLStarlet, a tool based on XSLT (Extensible Stylesheet Language Transform) (http://www.w3schools.com/xsl/). I will show you how to use this tool to quickly extract information out of the XML files.

The catch-22 of XSLT

XSLT, while a very powerful technique for manipulating XML, is itself written in XML. Therefore, to deal with the verbosity of XML, you still need to deal with this verbosity in XSLT.

All of this is changed by the XMLStarlet (http://xmlstar.sourceforge.net/), a multi-platform free software utility written by *Mikhail Grushinskiy* on top of the libxml2 and libxslt libraries. While XMLStarlet has the full functionality of the XSLT engine underneath, its interface is compact in the style of UNIX utilities, such as `grep` and `find`. I am assuming of course, that you find UNIX utilities simple; if you do not, the trade-off of compactness for clarity may not be worth it.

To me, XMLStarlet is *the* UNIX toolkit for the XML world as it is capable of queries (think `find` and `grep`), editing (`sed`) and other - more XML specific - operations such as validation, formatting and canonicalization.

Setting up

XMLStarlet is available for Linux, Solaris, MacOS X and Windows. I use it on Windows, but the examples should work the same way on all platforms. Please ensure that

XML Starlet's partial list of options

sel	Select data or query XML document(s) (XPATH, etc
ed	Edit/Update XML document(s)
tr	Transform XML document(s) using XSLT
val	Validate XML document(s) (well-formed/DTD/XSD/RelaxNG)
fo	Format XML document(s)
el	Display element structure of XML document

XMLStarlet's binary (xml or xml.exe) is on the path before any other programs of the same name. This is based on the program version 1.0.0.

Once you have XMLStarlet installed following the instructions bundled, try running the basic command `xml`. It should print out a long list of options.

A simple XML file

I will be using a basic phone book XML example to show the abilities of XMLStarlet. This could be something that your mobile phone would store or export its list of contacts to.

In the list, there are 3 contacts, each containing a person's name and a couple of his or her phone numbers. In real life, you would probably have several dozen contacts, each with more field types. But even with this simple example, you can probably see that it could be difficult to locate the information hidden within all the tags.

Sample phonebook XML file

```
<?xml version="1.0"?>
<phonebook>
 <contact>
  <name>John Doe</name>
  <phone type="home">555-1234</phone>
  <phone type="work">555-9876</phone>
 </contact>
 <contact>
  <name>Chris Jones</name>
  <phone type="work">555-9876</phone>
  <phone type="home">555-4567</phone>
  <phone type="mobile">555-5555</phone>
 </contact>
 <contact>
  <name>Jane Exciting</name>
  <phone type="home">555-1234</phone>
  <phone type="work">555-6543</phone>
 </contact>
</phonebook>
```

When you look at the information presented here, from inside the application owning the format, you most probably see one contact entry at a time, possibly even one phone number per screen.

XMLStarlet allows you to get below the proprietary interface and manipulate multiple entries in one go.

For this article, I will assume that the example XML is in the file phonebook.xml.

Validating the example XML

As a first step, I will validate the XML in the example above.

```
>xml val phonebook.xml
phonebook.xml - valid
```

This checks that the XML is well-formed. As I don't have a DTD or a schema supplied, this is all the validation I can do at the moment.

Had I mistyped the XML, I'd have received a detailed error message. For an example try removing the 'a' character on the second last line (the closing tag `</contact>`). You should get an error message similar to the following text:

```
phonebook.xml:18: parser error : Opening and ending
  tag mismatch:
contact line 14 and contct
  </contct>
      ^
phonebook.xml - invalid
```

As the message gives the start tag line (14), the problem line (18) and the actual error, it should be fairly easy to understand what went wrong and how to fix it.

While XMLStarlet has the full functionality of the XSLT engine underneath, its interface is compact in the style of UNIX utilities, such as `grep` and `find`

A basic query

Next, a very simple task: listing just the names of the people in our phone book.

```
> xml sel -t -m //contact -v name -n phonebook.xml
John Doe
Chris Jones
Jane Exciting
```

In this code, the `sel`(ect) tool, from the XMLStarlet's toolkit, is used and - this is where we start to see the power - so is XMLStarlet's own template language (`-t`). This particular expression (`xml sel -t`) puts XMLStarlet into the within-file find mode. To see all of the options available in the select mode, run the command `xml sel`.

The actual query is read as follows:

- `-m //contact` - Find element contact at any depth in the XML and for each such element...
- `-v name` - print the value of the child element called name...
- `-n` - ...and finish each printed match with a new line
- `phonebook.xml` - The file to run the query against

Extending the simple query

Once the basic query is running and the expected results can be seen, more options and conditions can be added to fine-tune the output.

In order to print the same list sorted by name, use the following code:

```
> xml sel -t -m //contact -s A:T:U name -v name -n phonebook.xml

Chris Jones
Jane Exciting
John Doe
```

The sort parameters come from the XSLT and mean alphabetically (`A`) as text (`T`) with upper case before lower (`U`).

What if you only wanted to print people who have mobile phones?

```
> xml sel -t -m //contact[phone/@type='mobile'] -v name -o '':''  -v phone[@type='mobile'] -n phonebook.xml
Chris Jones:555-5555
```

The square brackets hold conditions and use XPath expressions (`http://www.w3schools.com/xpath/default.asp`) to search through children nodes of the `contact` to find one called `phone` that will have its attribute `type` equal to "mobile".

A more complex query

Now, to build on previous queries and actually print all of the phone numbers as well as the names.

```
> xml sel -t
 -m //contact -s A:T:U name
 -v name -n
 -m phone -s A:T:U @type
 -v "concat('  ',@type,': ',self::phone)" -n
 -b
 -o "----" -n
 phonebook.xml
Chris Jones
  home: 555-4567
  mobile: 555-5555
  work: 555-9876
----
Jane Exciting
  home: 555-1234
  work: 555-6543
----
John Doe
  home: 555-1234
  work: 555-9876
----
```

Now to deconstruct and analyse this query step by step:

- `-m //contact -s A:T:U name` - Find each contact (sorted) and...
- `-v name` - print its name followed by the new line and then...
- `-m phone -s A:T:U @type` - ...for each phone of that contact (sorted by phone type).
- `-v ''concat('  ',@type,': ',self::phone)'' -n` - Print a line with leading spaces, phone type and the phone itself
- `-b` - We are done with the phones. -b terminates the scope of the nearest -m, in this case -m phone
- `-o ''----'' -n` - Finally, after all of the phones are printed for that contact, print a separating dashed line

It's obvious a fairly long nested query can be built incrementally until the output satisfies the need. And as the expression language is fairly compact, the cost of rewriting the template is fairly small.

Under the hood

If you have reached the limit of XMLStarlet's expression language or need more advanced functionality, like variables or keys, you can easily convert your query into XSLT and continue to use its full expressive power.

XMLStarlet is *the* UNIX toolkit for the XML world as it is capable of queries (think `find` and `grep`), editing (`sed`) and other - more XML specific - operations such as validation, formatting and canonicalization

---

To produce the XSLT equivalent of the XMLStarlet's query, all that is required is to add a -C flag in between `sel` and `-t`. So the last query becomes:

```
> xml sel -C -t -m //contact -s A:T:U name -v name \
  -n -m phone -s A:T:U @type \
  -v "concat('  ',@type,': ',self::phone)" -n -b \
  -o "----" -n phonebook.xml > phonebook.xsl
```

The content of the phonebook.xsl file should be:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:exslt="http://exslt.org/common"

       (...10 more lines of namespaces removed...)

extension-element-prefixes="exslt math date
func set str dyn saxon xalanredirect xt libxslt test"
 exclude-result-prefixes="math str">
<xsl:output omit-xml-declaration="yes" indent="no"/>
<xsl:param name="inputFile">-</xsl:param>
<xsl:template match="/">
 <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
 <xsl:for-each select="//contact">
  <xsl:sort order="ascending" data-type="text"
       case-order="upper-first" select="name"/>
  <xsl:value-of select="name"/>
  <xsl:value-of select="'&#10;'"/>
  <xsl:for-each select="phone">
   <xsl:sort order="ascending" data-type="text"
        case-order="upper-first" select="@type"/>
   <xsl:value-of select=
        "concat('  ',@type,': ',self::phone)"/>
   <xsl:value-of select="'&#10;'"/>
  </xsl:for-each>
  <xsl:value-of select="'----'"/>
  <xsl:value-of select="'&#10;'"/>
 </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

As can be seen, one line of switches mushrooms out into 36 lines of XSLT. I have no desire or patience to write that much XSL code during the prototyping stage, even with the help of an auto-completing XML editor. XMLStarlet allows me to rough out the transformation code much faster.

A couple of things to note while you are trying to understand the direct mapping:

- The transformation code itself is inside the template `t1`. Most of the lines before that are the static portion of an XSLT file - these will be the same for any query you run
- You don't need most of the `xmlns:XXX` lines if you are just starting with XSLT/XMLStarlet; they are there for the advanced functions
- `-n` (newline) flag translates into `
` symbol; not an easy mapping to discover or remember
- `-m` flag not only translates into `xsl:for-each` statement, but also sets the scope (until the end or the matching `-b` flag)
- `-s` flag becomes a very verbose `xsl:sort` statement
- You can combine several `value-of` statements into one using `concat` statement in the long form; in the short form it is easier to use `-v -n -v` flag sequence

To run the resulting XSLT, you can use any XSLT engine or XMLStarlet itself as follows:

```
xml tr phonebook.xsl phonebook.xml
```

Chaining it all together

As you can see, XMLStarlet allows you to quickly prototype an idea and produce a nicely formatted output. However it doesn't mean that XMLStarlet is only useful for quick and dirty output. In a similar way to the groff pipeline,

XMLStarlet can also be used to produce full applications when chained with other specialized utilities.

I will create one such application in this section. It will take a list of the contacts and produce a graph that will show people who might be sharing a work or home phone number and therefore work or live together.

To produce the graph, I'll use another free multi-platform utility called GraphViz (http://www.graphviz.org/).

In its most basic form, the input to the GraphViz utility needs to look like this:

```
digraph graphname {
nodea->nodec;
nodeb->nodec;
"node d"->"node e"[label=edgename];
}
```

In this code, nodes `nodea` and `nodeb` both point to the `nodec`. Node "`node d`" points to the "`node e`" and their graph connection is labelled `edgename`. Notice that spaces and other special characters in the names and labels requiring the whole name to be quoted.

Now I want to produce this type of output using XMLStarlet:

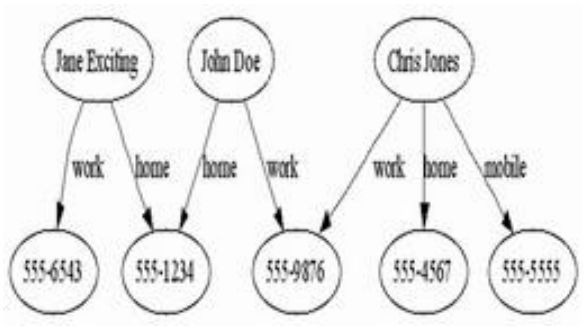
```
xml sel -T -t
 -o "digraph phonebook {" -n
 -m //contact/phone
 -v "concat(
    '&quot;',parent::contact/child::name,'&quot;')"
 -o "->"
 -v "concat('&quot;',self::phone,'&quot;')"
 -v "concat('[label=',@type,'];')" -n
 -b -o "}" -n phonebook.xml > phonebook.dot
```

Notice that `&quot;` will produce the double quote character. The strange looking `parent::contact/child::name` construct is another one of the XPath details which navigates us from the current context - which is `phone` - to its parent `contact` node and then reverses direction and finds the child node `name`. Basically, it finds the name of the person that the phone number belongs to. It would be slightly easier to do this in full XSLT with variables, but then we would lose our prototyping speed.

Running the above command line against our sample code will result in the following output in the phonebook.dot file:

```
digraph phonebook {
"John Doe"->"555-1234"[label=home];
"John Doe"->"555-9876"[label=work];
"Chris Jones"->"555-9876"[label=work];
"Chris Jones"->"555-4567"[label=home];
"Chris Jones"->"555-5555"[label=mobile];
"Jane Exciting"->"555-1234"[label=home];
"Jane Exciting"->"555-6543"[label=work];
}
```

GraphViz output of people's relationships



There should be no surprises here after the examples already done. Now run this through the GraphViz:

```
dot -Tjpg -ophonebook.jpg phonebook.dot
```

In here, `dot` is one of several GraphViz utilities and is used to produce directed hierarchical graphs.

As you can see from the graph, John Doe works together with Chris Jones and lives with Jane Exciting. Locating these relationships in the original XML file would take much more time than the single glance required with the GraphViz's output. And the difference would become even more pronounced with dozens of entries.

Going forward with XMLStarlet

In this article, I've only looked at XMLStarlet's query capability. That's how I started with the tool and with XSLT itself. Once the *query* interface is understood and XPath is no longer a stranger, it is worth having a look at *edit* options. Also the *format* options will come in handy whenever you have badly formatted XML and want to make it more human readable. And if you need to do something with XML that you think will take a long time, review what XMLStarlet provides and you may find a way to shortcut your solution process. It worked that way for me. Finally,

the documentation that comes with the tool has a number of detailed examples that may explain the finer points of the interface and the command set. Reading through it will increase your understanding of the tool and - most probably - XSLT itself.

Conclusion

XML is a very popular format with many benefits due to its structure. However, it is fairly verbose and is labour-intensive to manipulate. XMLStarlet changes all of this by bringing Unix-style toolkit capabilities to the manipulation and extraction of information from the XML. Combined with other utilities such as GraphViz, XMLStarlet allows for rapid application prototyping and development with a high return on time invested.

Bibliography

XMLStarlet (http://xmlstar.sourceforge.net/)

GraphViz (http://www.graphviz.org/)

Introduction to XPath (http://www.w3schools.com/xpath/)

Introduction to XSLT (http://www.w3schools.com/xsl/)

Copyright information

About the author

Alexandre Rafalovitch is a developer with more than 15 years of IT experience. He specializes in Java and XML technology. Alexandre has worked for several years as BEA senior technical support engineer, which gave him a strong impetus to study *quick solution* tools.