The xmllint Shell

Peter Lavin

2013-06-15

Table of Contents

XML files are human-readable, text files so it is easy to search them from the command line using grep or from within a text editor. But if you want to do something a little more sophisticated—count the number of elements, for example—you'll need to take a different approach. You could write a transformation style sheet to extract such information but this would be overkill. It is much easier to use **xmllint** from the command line to find out this kind of information.

This command is available on Mac OS X and Linux. It is installed by default on Mac OS X and, on Linux, if it isn't already installed, you can quickly do so by installing the `libxml2` package.

1. xmllint Options

One of the primary uses for the **xmllint** command is to validate that an XML file is well formed and that it conforms to a specific DTD or schema; this is done by using the `--valid` option. If your XML file contains other XIncluded files you can also use xmllint in the following way to resolve included files and output the result to a file:

```
shell> xmllint --xinclude manual.xml --output tmp.xml
```

The output file `tmp.xml` will include the contents of any `xi:include` elements. Also, the `--format` option is very useful for quickly formatting files from the command line. However, the most interesting option is the `--shell` option.

*For a complete list of all the options available view the **xmllint** man page.*

2. The xmllint Shell

Use xmllint with the `--shell` option in the following way:

```
shell> xmllint --shell file_name.xml
```

You can use other options with the `--shell` option. For example, if you wish to resolve included files, use the `--xinclude` option as well.

You can display the list of the commands available from the shell by typing `help`. You should see output similar to the following:

```
  base         display XML base of the node
  setbase URI  change the XML base of the node
  bye          leave shell
  cat [node]   display node or current node
  cd [path]    change directory to path or to root
```

```
  dir [path]   dumps informations about the node
               (namespace, attributes, content)
  du [path]    show the structure of the subtree under
               path or the current node
  exit         leave shell
  help         display this help
  free         display memory usage
  load [name]  load a new document with name
  ls [path]    list contents of path or the current directory
  set xml_fragment replace the current node content with the
               fragment parsed in context
  xpath expr   evaluate the XPath expression in that context
               and print the result
  setns nsreg  register a namespace to a prefix in the
               XPath evaluation context
               format for nsreg is: prefix=[nsuri]
               (i.e. prefix= unsets a prefix)
  setrootns    register all namespace found on the
               root element the default namespace
               if any uses 'defaultns' prefix
  pwd          display current working directory
  quit         leave shell
  save [name]  save this document to name or the original name
  write [name] write the current node to the filename
  validate     check the document for errors
  relaxng rng  validate the document against the Relax-NG schemas
  grep string  search for a string in the subtree
```

There are a number of relatively trivial but necessary commands such as **help** and **exit**. All the commands are useful but this article deals primarily with the following commands:

- `cat` *`node`* – output all nodes below the current node
- `cd` *`path`* – change to another node; you can only use this command with unique nodes.
- `dir` – dump information about the current node
- `xpath` *`expression`* – evaluate and print the XPath expression
- `setns` – register a namespace
- `write` *`filename`* – write the current node to file

*If you want to write your complete shell session to file run the shell after first issuing the **script** command. This can be particularly useful on Mac OS X where the **write** command does not work.*

3. Using Shell Commands

When you first open the xmllint shell the cursor, `/ >`, indicates that you are at the root node. You will likely want to navigate to specific nodes and view the file contents below that node. You can do this with the **cd** and **cat** commands.

```
/ > cd /options/option[@name = 'address_metrics_lifetime']
option >
```

On success the cursor changes to the name of the current node. To view the current node, use the **cat** command—this displays output to the screen. To create a text file of the output of cat, use **`write`** ***`file_name`*****`.xml`**.

You can only use **cd** to navigate to unique nodes. Attempt to navigate to a non-unique node and you will see output such as the following:

```
/ > cd /options/option
/options/option is a 353 Node Set
```

If there is no unique identifier for the node that you wish to navigate to, you can use a subscript in the following way:

```
/ > cd /options/option[1]
option >
```

To output information about the current node use the **dir** command:

```
option > dir
ELEMENT option
  ATTRIBUTE name
    TEXT
      content=address_metrics_cleanse_interval
  ATTRIBUTE type
    TEXT
      content=sending
option >
```

4. Working with Multiple Files

You can open the xmllint shell specifying multiple files but the behaviour is not intuitive. In the following example, the shell is opened with two different files that have the same structure. The `options.xml` has a root element `<options>` with 353 `<option>`s while the `smpp_options.xml` has a root element `<options>` containing only 57 `<option>`s.

```
shell> xmllint --shell options.xml smpp_options.xml
/ > base
options.xml
/ > xpath count(//option)
Object is a number : 353
/ > bye
/ > base
smpp_options.xml
/ > xpath count(//option)
Object is a number : 57
/ > setbase options.xml
/ > base
options.xml
```

If you invoke `help` from the shell the **bye** command is tersely described as `leave shell`. As this sequence of commands shows, `bye` also exits the first file passed to the `--shell` option.

Once you have exited the first shell, you cannot return to it by using `setbase` even though the command seems to have performed it's function—as the output of `base` erroneously indicates. For this reason it is perhaps less confusing to open the shell specifying only one file and then use the load command to switch to a different file:

```
shell> xmllint --shell options.xml
/ > base
options.xml
/ > xpath count(//option)
Object is a number : 353
/ > load smpp_options.xml
/ > base
smpp_options.xml
/ > xpath count(//option)
Object is a number : 57
```

The second count indicates that the load command executed successfully.

5. Using Namespaces

To this point none of the examples use namespaces. To use an XML file with namespaces you must use the **setns** command. Use it in the following way:

```
shell> xmllint --xinclude --shell manual.xml
/ > setns x=http://docbook.org/ns/docbook
/ > dir
DOCUMENT
version=1.0
URL=manual.xml
standalone=true
namespace xml href=http://www.w3.org/XML/1998/namespace
/ > cd /x:book/x:chapter[@xml:id='apis']
chapter > dir
ELEMENT chapter
  ATTRIBUTE id
    TEXT
      content=apis
```

The **dir** command shown above confirms that you have navigated to the specified node. From that node you can execute **xpath** commands using absolute or relative paths.

```
chapter > xpath count(/x:book/x:chapter[@xml:id='apis']/x:section)
Object is a number : 15
chapter > xpath count(/x:book/x:chapter[@xml:id='apis']/x:section/x:refentry)
Object is a number : 135
chapter > xpath count(/x:book/x:chapter[@xml:id='structs']/x:section/x:section)
Object is a number : 18
chapter > xpath count(//x:chapter[@xml:id='apis']/x:section/x:refentry)
Object is a number : 135
chapter > xpath count(//x:section/x:refentry)
Object is a number : 140
chapter > xpath count(x:section/x:refentry)
Object is a number : 135
```

There are 15 sections in the `apis` chapter and these 15 sections have 135 refentries. Note the difference in output between the paths `//x:section/x:refentry` and `x:section/x:refentry`. The difference in output shows that only the latter is relative to the current node.

When your XML file uses IDs, an easier way to navigate is to use the `id` function:

```
 chapter > cd /
/ > xpath id('apis')
Object is a Node Set :
Set contains 1 nodes:
1  ELEMENT chapter
    ATTRIBUTE id
      TEXT
        content=apis
/ > cd id('apis')
chapter > xpath count(x:section/x:refentry)
Object is a number : 135
chapter > cd /
/ > xpath count(id('apis')/x:section/x:refentry)
Object is a number : 135
```

For files that use namespaces, you must set the namespace before you can use the `id` function.

As indicated above, the `id` function can also be used inside the `count` function.

6. About the Author

Peter Lavin is a technical writer who has been published in a number of print and online magazines. He is the author of Object Oriented PHP, published by No Starch Press and a contributor to PHP Hacks by O'Reilly Media.