xmlsh

A command line shell for XML
Based on the philosophy
and design of the
Unix Shells

David A. Lee
dlee@calldei.com

www.xmlsh.org

Overview

- Motivation
- Project
- Philosophy
- Syntax and Features
- Architecture
- Demonstration & Examples
- Roadmap
- Contribute
- Feedback / Q&A

Motivation

- Unix and Unix Shells were a radical “Paradigm Shift”
  - Vastly simplified access to data and processing
  - Set of small simple core tools
  - Create complexity with hierarchy instead of linearly.
- Almost 40 years later and the core design fundamentals are being eroded
  - Predominant data type is no longer byte/line streams
  - Tools and shells have not evolved with the data (XML).
  - Working with XML is way too complicated !

Project

- Open Source Project
  - Open Source / Closed Development
  - BSD License "Free Software"
  - No commercial restrictions
  - Hosted on Sourceforge: **xmlsh.sourceforge.net**
  - Main project site: **www.xmlsh.org**
- Currently "Pre Alpha"
  - Ready for experimentation – ***not for production***
  - Syntax subject to change
  - Internal API's subject to change

Project

- Pure Java
  - Tested on Windows, Mac and Linux
  - Should run on any OS that runs Java 1.6
- Dependencies
  - Saxon 9
  - Log4J
  - Optional external OS commands
    (rm , chmod , date … haven't re-invented the wheel)

Unix Shells - Philosophy

- What's ***Great*** about the Unix Shells
  - Thrived almost 40 years and many incarnations
  - Ideal balance between CLI and Programming language

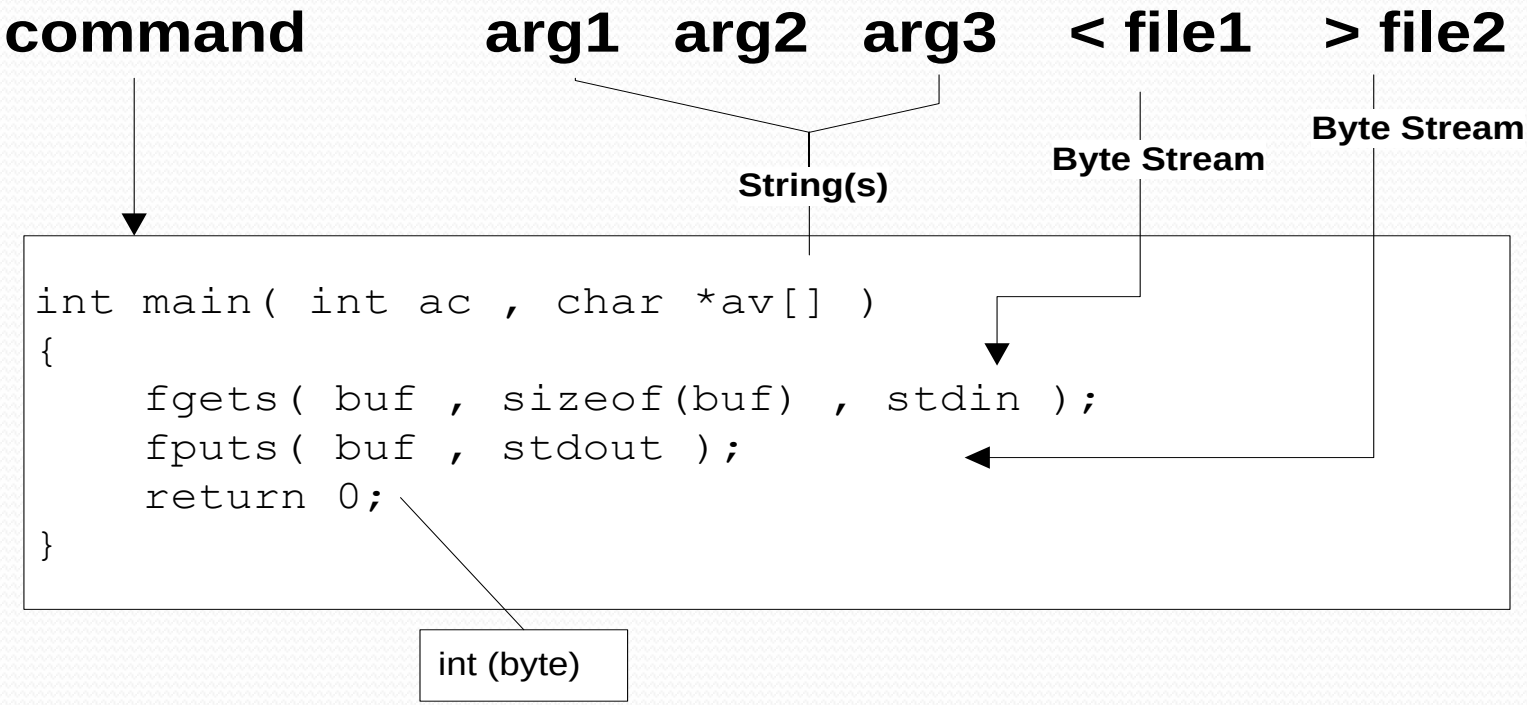
"Although most users think of the shell as an interactive command interpreter, it is really a programming language in which each statement runs a command. Because it must satisfy both the interactive and programming aspects of command execution, it is a strange language, shaped as much by history as by design."

– Brian Kernighan & Rob Pike ,
The UNIX Programming Environment", Prentice-Hall (1984).

Unix Shells – Philosophy

- What's ***Great*** about the Unix Shells
  - All IO is byte streams (or text line streams)
    - Core toolkit designed around a simple universal type
      byte or line streams
      - ***wc cat ls sed grep cut paste head read tail awk more*** …
    - All files and devices are byte/line streams.
    - programs consume and produce byte/line streams.
  - Core "toolkit" of simple components
  - Enables creation of complexity through hierarchal combination of simpler components.

Anatomy of a Unix Shell command



Unix Shells – Philosophy

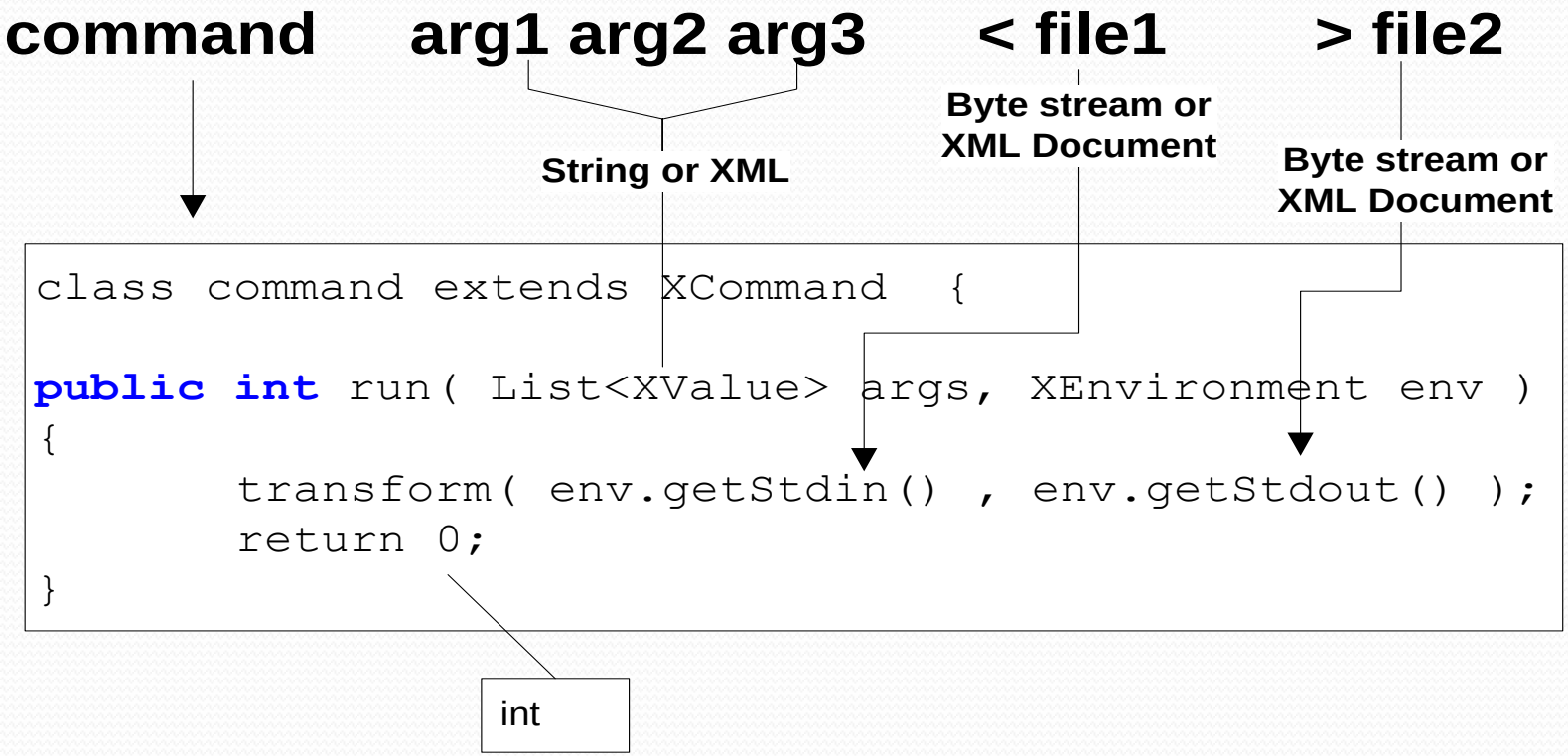
- What's ***Wrong*** with Unix Shells ?
  - Today's data is no longer primarily text (byte/line streams)
    Data is increasingly XML
  - Many core commands are not meaningful or
    dont work well with XML
    - wc cut grep paste tail head sed awk more cat …
- Why not just new commands ?
  - The shell itself is aging …
    made with the assumption that all data is strings or lines.
    - Flow control (for / case / read)
    - Variables / Environment / IO streams
    - Pipelines / Command input/output
  - Desire for a cross platform, portable shell

xmlsh – Philosophy

- Based on the design principles of the Unix Shells
- Largely backwards compatible syntax to /bin/sh
  - Use cases equivalent to /bin/sh
- Where the Unix Shells use strings and byte streams, xmlsh targets XML documents and infoset streams.
- ***Scripting with XML data should be as easy and natural as scripting with text files.***

- ***Someday, all data should be XML …***
  - But until then, intermixing Text and XML is necessary

Anatomy of a xmlsh command



Syntax and Features

- Core syntax equivalent to /bin/sh
  - if … then … else … fi
  - while/until … do …
  - case … in
  - functions
  - variable assignment
  - Pipes
  - subshells and background processes/threads
  - IO redirection
  - script and external process execution

Syntax and Features

- New syntax specific to xmlsh
  - XML expressions and variables

    <[*xquery expr*]>

    $<(*xml producing command*)

  Example:

    foo=<[“hi” , 123 , <elem attr=“attr”>body</elem>]>

    bar=$<(xls)

    for $i in <[1,2,3,<node/>,”hi”]> ; do echo $i ; done

Syntax and Features

- built-in commands similar to /bin/sh

:	exit
set	shift
source (.)	test ([*condition*])
true	false
read	xread
xwhich	xenv
echo	cd
jobs	wait

Syntax and Features

- internal commands (supplied with xmlsh)

xcat	**xcmp**
xls	xpwd
xpath	xslt
xquery	xsplit
xed	xversion
.... Many more to come ...	

xmlsh Features

- User commands
  - Can integrate directly to xmlsh and run within the same JVM and participate in internal architecture.
- External Commands
  - Can execute any external command supported by the OS
  - Can pipe into and out of external commands, same as internal commands.

Architecture

- Source is 100% pure Java using JDK 1.6
- Parser implemented with javacc
- Logging via log4j
- XSLT and XQuery from Saxon
  - XQuery heavily used internally

Architecture

- Variables
  - Dynamically typed variables (text , xml)
  - Take on the type of their assignment expression
    - x=“foo” # string
    - x=<[“foo”]> # xml
  - XML type is really a saxon “XdmValue”
    - atomic type
    - item type
    - sequence
    - Anything that XQuery can produce

Architecture

- Pipes
  - Pipeline commands run as separate threads
  - Pipe is a *Currently* a text pipe
    (XML is text serialized)
    - ***Future*** *– XML native pipes – binary or event serialized*
      Suggestions Welcome !
  - Internal/builtin commands in separate threads
  - External commands in separate processes

Architecture

- Built-in, Internal, and User commands run in the JVM
  - Access to native representations for shell environment
  - Access to same runtime (saxon, log4j etc)
  - XML data held as Saxon trees not text
  - Participates in multithreading
    - Background threads (cmd &)
    - Piping (cmd | cmd | cmd)
  - Arguments and variables passed in internal form (not converted to text)

Architecture

- External commands
  - May run any OS command (cp mv ls chmod gcc …)
    - xmlsh is not a replacement for the OS layer or commands
    - External Commands run as a sub process
  - Piping to and from external command
    Freely intermix internal and external commands
    - xquery | sum
    - xls | cat | xcat

Problems and Limitations

- *Javacc vs yacc, LL(1) vs LALR(1)*
  - *POSIX sh specs are LALR, challenging to translate to LL*
  - *Some syntax difficult to implement easily/correctly*
- Java runtime instead of Unix OS layer
  - Threading instead of processes
  - No real concept of File Descriptor (numbers like 0,1,2,3)
  - Console IO is limited
    - Cannot run console sub-processes which share stdin.
    - No good tty interrupts

Examples

- *Basic sh-like syntax*

```
dir=/output
for file in *.xml ; do
      xquery -f /path/pass1.xquery -i $file  |
      xslt  -f /path/pass2.xsl -i  > $dir/$file
done
```

- *Simple xml tools*

```
xcat *.xml | xpath '//book[@author="John Doe"']
```

Examples

- *XML and text Variables*
- *XML construction syntax (XQuery based)*

```
A= "text"

XVAR=<[
<foo attr= "bar">
      {$A}
</foo>
]>

echo $XVAR > file.xml
```

Examples

- *Reusable parsed XML Documents*
- *XML Sequences in for loops*

```
xread doc < file.xml

for i in <[for $x in 1 to 1000 return $x]> ; do
   xquery -i $doc '//part[@num=$i]' > out${i}.xml
   xpath -i $doc '//part[@num=$i]/@title' >> titles.txt
done
```

Roadmap

- Currently Pre-Alpha – Started Nov 2007
- To go to alpha …
  - Resolve core syntax questions
    - *Example:* should $* be a string or sequence ?
    - *Example:* should echo produce XML or Text ?
  - Clarify Philosophy
    - What xmlsh Is and Is Not – define scope
    - Clarify use cases
  - Zero defects in current codebase
    - Complete test cases
  - Solicit peer comments

Contribute

- Try it out !
- Report Bugs
- Discuss / Forums
  - Use cases
  - Design Discussions
  - Enhancements
  - What **do** you like ? Why ?
  - What **don't** you like ? Why ?

Feedback / Q&A / Demo