Perl vs. Forth

Peter Knaggs

University of Paisley, High Street,
Paisley. PA1 2BE Scotland.

pjk@bcs.org.uk

May 19, 1998

Abstract

"Perl" is a scripting language originally developed for the Unix environment, collecting a number of useful Unix tool into a flexible interpreted C-like language. Over the past three or four years Perl has taken the Unix world by storm, and has become the standard language for Internet (Common Gateway Interface or CGI) programming.

This paper looks at some of the perceived advantages of Perl and investigates what, if any, lessons the Forth community can learn from this language.

1 Overview of Perl

> To those who merely like it, Perl is the *Practical Extraction and Report Language*. To those who love it, Perl is the *Pathologically Eclectic Rubbish Lister*. And to the minimalists in the crowed, Perl seems like a pointless exercise in redundancy.
>
> (Wall, Christiansen, and Schwartz 1996)

Perl was original developed by Larry Wall as a shell for the Unix system. Over the years Larry kept on adding new features to Perl as he required them. By version 4, and now 5, of the language Perl has progressed from a scripting tool to a full blown programming language incorporating many of the tools the Unix shell programmer finds invaluable (`find`, `grep`, `sed`, `awk`, etc.).

So, just what are the features of Perl that make it particularly useful for programming Unix systems and CGI in particular. They can be broken down into five main areas.

1.1 Data Structures

Perl provides a number of flexible and very useful data structures. The first of these is the '*list*'. This is a simple list of data objects, which may be accessed by position, i.e., a simple array or tuple. However, operators are also provided that allows the list to be treated as a stack or as a queue.

The second, and probably the most useful of the data structures, is the '*hash*', or associative array. This is a relational structure where any object can be used as the key to the relation and any object can be used as its value, this includes a list (or tuple), another hash or a single data item. This provides the programmer with a built in relational database.

The final data structure is oriented towards Perl's official rôle of a report language. The '*format*' allows the programmer to specify the form of a record on the output device, with particular fields being bound to specific variables. The `write` command is used to write a record to the output device in the given format. The system keeps track of the number of lines used, the number of pages used, etc. It even allows for the definition of a separate header format, which will be output at the start of every page.

1.2 Objects

Version 4 introduced a concept known as *packages*. A package was simply a separate name space, very similar in concept to Forth's wordlists. The idea was grossly misunderstood by most users. In version 5 the idea was converted into a full blown object handling system, complete with a definable interface, constructor, destructor, etc.

The system provides for a library of pre-defined objects. These objects are kept in source form and may be loaded dynamically as required. It is interesting to note the large number of library objects provided by the Perl community. These are stored in a central FTP archive (the Comprehensive Perl Archive Network or CPAN), which is supported by the community.

1.3 Regular Expressions

Probably the most powerful aspect of Perl is its string handling. This is primarily handled via an extended regular expression (regex) handling, or pattern matching system. Perl's regex is based on the standard Unix regex as used in `grep` etc., but has extended the system into an extremely powerful string parser.

Many of the search engines available over the Internet accept a "Perl regular expression" as their search string. Although it is becoming more common to offer the user a simpler interface and convert the request into a Perl regex for internal processing.

In JavaScript version 1.2 they have introduced a regex object, and extended the string object to include regex handling methods. In particular they have introduced:

match will attempt to match a regex with the string. For example:

```
m/(\d*)\w(\d*)\w(\d*)/
```

will match with three numbers separated by a single white space character. The parentheses indicate sections of the regex that should be remembered. These sections are copied into the variables $1, $2, and $3 respectively.

replace will replace the matched expression with a new value, note that the $ variables are available for use in the replacement value. For example:

```
s/bold\(([^\)]*)\)/<B>$1</B>/
```

will replace any occurrence of "`bold(text)`" with "`<B>text</B>`".

split will return a list (or array) of strings, splitting the original according to the regex. For example:

```
$str = "hello.this is:a test"
@list = split(/.|,|:/,$str)
```

will split the string into three: "`hello`", "`this is`", and "`a test`", the split occurring on a dot, comma, or colon as described the regex "`/.|,|:/`". The variable @list will contain the three fragments as a list (array, stack, queue, or tuple, depending on how you access the variable).

Given that it is possible to read a single line, a paragraph/record, or a whole file into a single string and then manipulate the string using these built in regex functions this provides a very fast and powerful search/replace facility. It has been shown that a Perl program that reads a file into a string and then performs a match operation is faster than the equivalent Unix `grep` (get regular expression) command!

1.4 System Interaction

As Perl was originally developed as a shell it has a very close interaction with the operating system.

commands can be executed and the result analysed. Where a feature is not already incorporated into Perl, it provides the programmer with the facility to invoke the relevant command. The input to the command, and the output from the command are under programmer control, thus the program can respond to messages from the external program.

signals or interrupts can be intercepted. Special handlers can be provided for all of the standard signals (interrupts) by simply assigning a function to a special array.

communication ports are accessed via a set of standard 'socket' functions.

multitasking is handled by the standard Unix process functions, namely, fork, kill, and pipe. All fully integrated into to the basic language.

In addition to these built in facilities, Perl offers the ability of binding a hash with a database file, using the standard dbm system. Thus accessing an entry in the hash will in fact be an access to the database file. Consequently a new command (`delete`) has been provided that allows the programmer to delete an entry from the hash, and thus the database.

1.5 Source Control

Larry Wall has kept absolute control over the development of the language. Others have implemented new features, which Larry has incorporated into the next release of Perl or not as he sees fit. This means that there is a well known standard base on which to develop the application and/or library.

The version number of the executing system is available to the programmer. A special command has been provided to allow programmers to specify which version their module was designed for. For example, if a module takes advantage of a feature added to Perl version 5.1 he can place the command:

```
requires 5.1
```

at the beginning of the program. If this command is executed in Perl 5.1 or later, this will return true, otherwise it will report an error message stating that the code requires Perl version 5.1 or later.

2 Forth Lessons

It is possible to see from the preceding that there are a number of lessons the Forth community can learn from Perl and it's community. These can be split into two main areas: major and additional lessons.

2.1 Major lessons

There are two major lessons that the Forth community can learn from the Perl experience. Both of these lessons are already known to the community, however, it has been very slow in addressing them and promoting a unified position.

2.1.1 Source Control

The ISO Forth (International Standards Organisation 1997) standard is a good step in this direction, however, is does not guarantee the existence of any particular word in the dictionary. It is simply not possible to provide someone with the source for a complex program and expect it to work on their system without first specifying all kinds of environmental dependencies that would put most people off even attempting.

There needs to be a way of providing a central bank of library code that will operate across different Forth platforms. A bank of source code libraries written under ISO Forth would only be a start. Such a code bank would require organisation and documentation, thus making the relevant library easy to find and use.

A centralised library, complete with a librarian is required. The librarian can check the quality of the modules and more importantly the documentation submitted and accept or refuse the module. Having accepted a module they can then classify the module. All of the modules should be provided on a freeware or shareware basis.

2.1.2 Objects

Many people have looked at adding object oriented programming to Forth, each developing their own extensions. It is time a standard was developed for these extensions. The world embraced object oriented programming over 12 years ago, perhaps it time Forth was redeveloped to integrate objects

into the kernel rather than as a bolt-on.

2.2 Additional lessons

There are a number of additional lessons that require the Major lessons be learned and acted upon before they can truly be addressed. As with the major lessons the community is already aware of these, but have yet to address them to the satisfaction of the computing community at large (programmers, software engineers, computer engineers, computer scientists).

2.2.1 Flexible Data Structures

Larry Wall chose to provide two extremely useful and very flexible data structures, namely the *list* and the *hash*. These relate very closely to the mathematical concepts of a *tuple* and *mapping* respectively, and may consequently lead to some very simple and effective programming.

Such data structures are simple enough to implement on an individual basis. Rather than every Forth programmer developing their own version, the community should provide flexible list/hash objects as a source level library. Thus, only one or two programmers (those who provide the library module) need be concerned with the development of the objects, while the rest of the community simply use them.

2.2.2 String Handling

Given a truly object oriented Forth system it would be possible to provide a "string" object, which includes all of the normal string handling functions: read line, length, substring, concatenate, character at, etc. Taking a leaf from JavaScript/1.2 it would be possible to extend such a string object to include regular expression handling: match, replace, split. This would provide Forth with the same string handling capabilities as the most up-to-date languages.

2.2.3 System Interaction

With the introduction of a number of special objects it would be possible for all Forth system to provide an interface to the underlying system. These objects will have to be defined with grate care to allow interaction with large operating systems hosted systems and small single board systems. The device tree specified in the Boot Firmware (IEEE 1994) system is an example of how object oriented thinking can provide such a facility.

3 Conclusions

Can the Forth community learn any lessons from modern developments and methods? The forgoing indicates that there is much the community can learn from just one of these modern developments. The Forth community has many highly intelligent individuals, who are more than capable of learning these lessons. Unfortunately half of them are too interested in the internal workings of * while others are all 'individuals'.

In 1983 Forth was ahead of its time (Rather, Colburn, and Moore 1993) and with the Forth Interest Group to promote it, in a very strong position. The rest of the programming world caught up around 5 years later, now another 9 years later the world has moved on and Forth has hardly moved. In 1994 there was the ANS Forth (American National Standards Institute and Computer and Business Equipment Manufacturers Association 1994), later ratified as ISO Forth (International Standards Organisation 1997) in 1997, but the new standard is, while very useful, not going to drive the language forward, only the community can do that!

Nineteen ninety four also saw the introduction of the IEEE Open Boot Firmware standard (IEEE 1994), which did at least take the language further. If the Forth language is to be a programming language of the future it is going to have to catch up with modern systems, and ideas about programming. Otherwise, it is going to become another interesting relic.

Some kind of central authority is need to push the language and further its development. The Forth

Interest Group has been behind the language for many years, yet there has been very little development evident.

References

American National Standards Institute and Computer and Business Equipment Manufacturers Association (1994, March). *American National Standard for information systems: programming languages: Forth: ANSI/X3.215-1994*. 1430 Broadway, New York, NY 10018, USA: American National Standards Institute. Approved March 24, 1994.

IEEE (1994). *IEEE Standard 1275-1994 — Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices*. IEEE. IEEE 1275 Technical Committee.

International Standards Organisation (1997, March). *Information technology — Programming languages — Forth* (First ed.). International Standards Organisation. ISO/IEC 15145:1997.

Rather, E. D., D. R. Colburn, and C. H. Moore (1993, March). The evolution of Forth. In ACM (Ed.), *ACM SIGPLAN HOPL-II. 2nd ACM SIGPLAN History of Programming Languages Conference (Preprints)*, Volume 28(3) of *ACM SIGPLAN Notices*, New York, NY, USA, pp. 177–199. ACM Press.

Wall, L., T. Christiansen, and R. L. Schwartz (1996). *Programming Perl* (Second ed.). O'Reilly and Associates, Inc.