Research Paper On

Perl

(High-level Programming Language)



University of Computer Studies (Pathein)

Third Year, Section(A)

2017-2018 Academic Year

Supervised by: Daw Khin Lay Thwin

Presented by:

Mg Zay Min Paing 3CS-3

Mg Aung Myo Than 3CS-24

Ma Wut Yee Kyaw 3CS-5

Ma Yadanar Hlaing 3CS-10

Ma Hnin Yu Wai 3CS-18

Ma Nay Zin Oo 3CS-25

Submitted September, 2018

**Introduction**

**Perl** is a family of two high-level, general-purpose, interpreted, dynamic programming languages.

Though Perl is not officially an acronym, there are various backronyms in use, including "Practical Extraction and Reporting Language". Perl was originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language to make report processing easier. Since then, it has undergone many changes and revisions. Perl 6, which began as a redesign of Perl 5 in 2000, eventually evolved into a separate language. Both languages continue to be developed independently by different development teams and liberally borrow ideas from one another.

The Perl languages borrow features from other programming languages including C, shell script (sh), AWK, and sed; Wall also alludes to Basic and Lisp in the introduction to *Learning Perl* (Schwartz & Christiansen) and so on. They provide text processing facilities without the arbitrary data-length limits of many contemporary Unix commandline tools, facilitating manipulation of text files. Perl 5 gained widespread popularity in the late 1990s as a CGI scripting language, in part due to its then unsurpassed regular expression and string parsing abilities.

In addition to CGI, Perl 5 is used for system administration, network programming, finance, bioinformatics, and other applications, such as for GUIs. It has been nicknamed "the Swiss Army chainsaw of scripting languages" because of its flexibility and power, and also its ugliness. In 1998, it was also referred to as the "duct tape that holds the Internet together", in reference to both its ubiquitous use as a glue language and its perceived inelegance.

**History**

**Early Versions**

Larry Wall began work on Perl in 1987, while working as a programmer at Unisys, and released version 1.0 to the comp.sources.misc newsgroup on December 18, 1987. The language expanded rapidly over the next few years.

Perl 2, released in 1988, featured a better regular expression engine. Perl 3, released in 1989, added support for binary data streams.

Originally, the only documentation for Perl was a single lengthy man page. In 1991, *Programming Perl*, known to many Perl programmers as the "Camel Book" because of its cover, was published and became the *de facto* reference for the language. At the same time, the Perl version number was bumped to 4, not to mark a major change in the language but to identify the version that was well documented by the book.

**Early Perl 5**

Perl 4 went through a series of maintenance releases, culminating in Perl 4.036 in 1993. At that point, Wall abandoned Perl 4 to begin work on Perl 5. Initial design of Perl 5 continued into 1994. The *perl5-porters* mailing list was established in May 1994 to coordinate work on porting Perl 5 to different platforms. It remains the primary forum for development, maintenance, and porting of Perl 5.

Perl 5.000 was released on October 17, 1994. It was a nearly complete rewrite of the interpreter, and it added many new features to the language, including objects, references, lexical (my) variables, and modules. Importantly, modules provided a mechanism for extending the language without modifying the interpreter. This allowed the core interpreter to stabilize, even as it enabled ordinary Perl programmers to add new language features. Perl 5 has been in active development since then.

Perl 5.001 was released on March 13, 1995. Perl 5.002 was released on February 29, 1996 with the new prototypes feature. This allowed module authors to make subroutines that behaved like Perl builtins. Perl 5.003 was released June 25, 1996, as a security release.

One of the most important events in Perl 5 history took place outside of the language proper and was a consequence of its module support. On October 26, 1995, the Comprehensive Perl Archive Network (CPAN) was established as a repository for Perl modules and Perl itself; as of May 2017, it carries over 185,178 modules in 35,190 distributions, written by more than 13,071 authors, and is mirrored worldwide at more than 245 locations.

Perl 5.004 was released on May 15, 1997, and included among other things the UNIVERSAL package, giving Perl a base object to which all classes were automatically derived and the ability to require versions of modules. Another significant development was the inclusion of the CGI.pm module, which contributed to Perl's popularity as a CGI scripting language.

Perl is also now supported running under Microsoft Windows and several other operating systems.

Perl 5.005 was released on July 22, 1998. This release included several enhancements to the regex engine, new hooks into the backend through the B::\* modules, the qr// regex quote operator, a large selection of other new core modules, and added support for several more operating systems, including BeOS.

**2000–present**

Perl 5.6 was released on March 22, 2000. Major changes included 64-bit support, Unicode string representation, support for files over 2 GiB, and the "our" keyword. When developing Perl 5.6, the decision was made to switch the versioning scheme to one more similar to other open source projects; after 5.005\_63, the next version became 5.5.640, with plans for development versions to have odd numbers and stable versions to have even numbers.

In 2000, Wall put forth a call for suggestions for a new version of Perl from the community. The process resulted in 361 RFC (request for comments) documents that were to be used in guiding development of Perl 6. In 2001, work began on the "Apocalypses" for Perl 6, a series of documents meant to summarize the change requests and present the design of the next generation of Perl. They were presented as a digest of the RFCs, rather than a formal document. At this point, Perl 6 existed only as a description of a language.

Perl 5.8 was first released on July 18, 2002, and had nearly yearly updates since then. Perl 5.8 improved Unicode support, added a new I/O implementation, added a new thread implementation, improved numeric accuracy, and added several new modules. As of 2013 this version still remains the most popular version of Perl and is used by Red Hat 5, Suse 10, Solaris 10, HP-UX 11.31 and AIX 5.

In 2004, work began on the "Synopses" – documents that originally summarized the Apocalypses, but which became the specification for the Perl 6 language. In February 2005, Audrey Tang began work on Pugs, a Perl 6 interpreter written in Haskell. This was the first concerted effort towards making Perl 6 a reality. This effort stalled in 2006.

On December 18, 2007, the 20th anniversary of Perl 1.0, Perl 5.10.0 was released. Perl 5.10.0 included notable new features, which brought it closer to Perl 6. These included a switch statement (called "given"/"when"), regular expressions updates, and the smart match operator, "~~". Around this same time, development began in earnest on another implementation of Perl 6 known as Rakudo Perl, developed in tandem with the Parrot virtual machine. As of November 2009, Rakudo Perl has had regular monthly releases and now is the most complete implementation of Per 6.

A major change in the development process of Perl 5 occurred with Perl 5.11; the development community has switched to a monthly release cycle of development releases, with a yearly schedule of stable releases. By that plan, bugfix point releases will follow the stable releases every three months.

On April 12, 2010, Perl 5.12.0 was released. Notable core enhancements include new package NAME VERSION syntax, the Yada Yada operator (intended to mark placeholder code that is not yet implemented), implicit strictures, full Y2038 compliance, regex conversion overloading, DTrace support, and Unicode 5.2. On January 21, 2011, Perl 5.12.3 was released; it contains updated modules and some documentation changes. Version 5.12.4 was released on June 20, 2011. The latest version of that branch, 5.12.5, was released on November 10, 2012.

On May 14, 2011, Perl 5.14 was released. JSON support is built-in as of 5.14.0. The latest version of that branch, 5.14.4, was released on March 10, 2013.

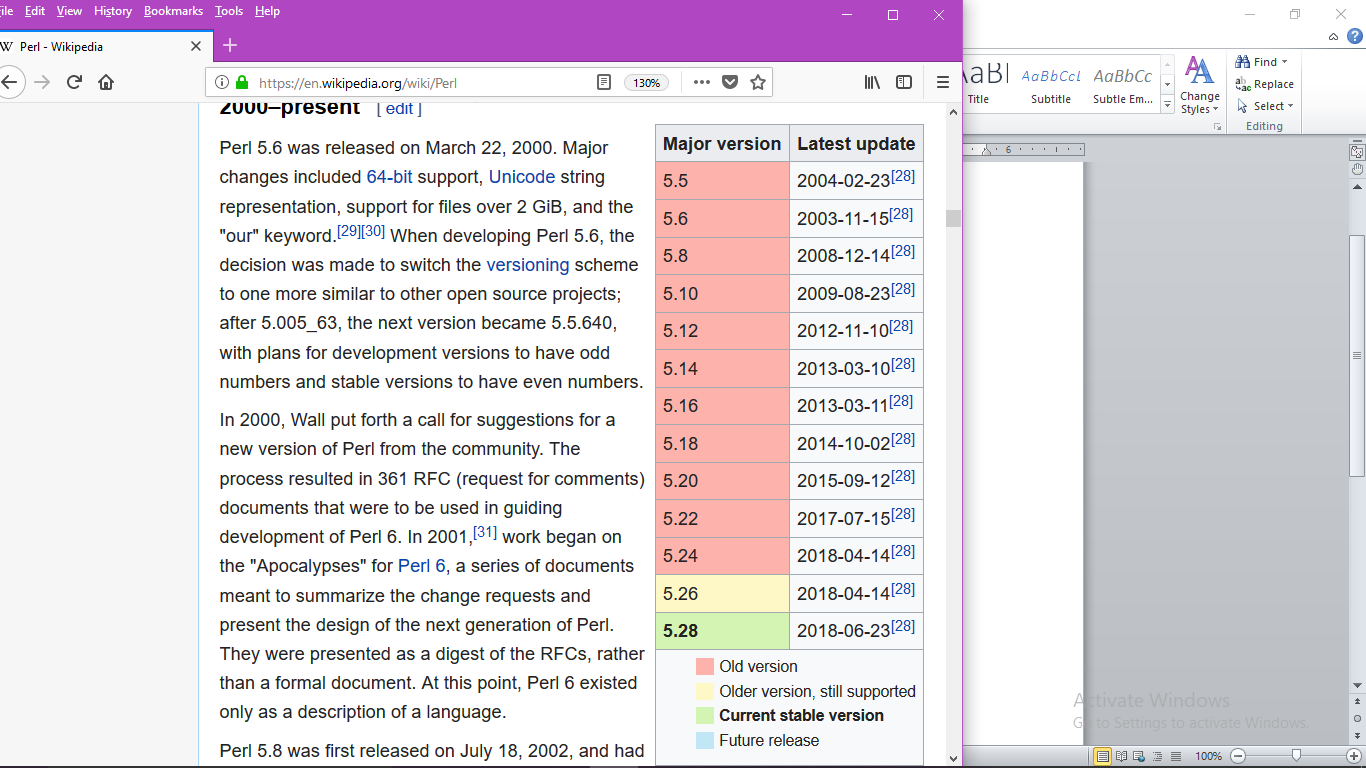
On May 20, 2012, Perl 5.16 was released. Notable new features include the ability to specify a given version of Perl that one wishes to emulate, allowing users to upgrade their version of Perl, but still run old scripts that would normally be incompatible. Perl 5.16 also updates the core to support Unicode 6.1.

On May 18, 2013, Perl 5.18 was released. Notable new features include the new dtrace hooks, lexical subs, more CORE:: subs, overhaul of the hash for security reasons, support for Unicode 6.2.

On May 27, 2014, Perl 5.20 was released. Notable new features include subroutine signatures, hash slices/new slice syntax, postfix dereferencing (experimental), Unicode 6.3, rand() using consistent random number generator.

Some observers credit the release of Perl 5.10 with the start of the Modern Perl movement. In particular, this phrase describes a style of development that embraces the use of the CPAN, takes advantage of recent developments in the language, and is rigorous about creating high quality code. While the book "Modern Perl" may be the most visible standard-bearer of this idea, other groups such as the Enlightened Perl Organization have taken up the cause.

In late 2012 and 2013, several projects for alternative implementations for Perl 5 started: Perl5 in Perl6 by the Rakudo Perl team, *moe* by Stevan Little and friends, *p2* by the Perl11 team under Reini Urban, *gperl* by goccy, and *rperl* a kickstarter project led by Will Braswell and affiliated with the Perll11 project.



### PONIE

**PONIE** is an acronym for Perl On New Internal Engine. The PONIE Project existed from 2003 until 2006 and was to be a bridge between Perl 5 and Perl 6. It was an effort to rewrite the Perl 5 interpreter to run on Parrot, the Perl 6 virtual machine. The goal was to ensure the future of the millions of lines of Perl 5 code at thousands of companies around the world.

The PONIE project ended in 2006 and is no longer being actively developed. Some of the improvements made to the Perl 5 interpreter as part of PONIE were folded into that project.

### Name

Perl was originally named "Pearl". Wall wanted to give the language a short name with positive connotations; he claims that he considered every three- and four-letter word in the dictionary. He also considered naming it after his wife Gloria. Wall discovered the existing PEARL programming language before Perl's official release and changed the spelling of the name.

When referring to the language, the name is normally capitalized (*Perl*) as a proper noun. When referring to the interpreter program itself, the name is often uncapitalized (*perl*) because most Unix-like file systems are case-sensitive. Before the release of the first edition of *Programming Perl*, it was common to refer to the language as *perl*; Randal L. Schwartz, however, capitalized the language's name in the book to make it stand out better when typeset. This case distinction was subsequently documented as canonical.

The name is occasionally expanded as *Practical Extraction and Report Language*, but this is a backronym. Other expansions have been suggested as equally canonical, including Wall's own *Pathologically Eclectic Rubbish Lister* which is in the manual page for perl. Indeed, Wall claims that the name was intended to inspire many different expansions.

### Camel symbol

*Programming Perl*, published by O'Reilly Media, features a picture of a dromedary camel on the cover and is commonly called the "Camel Book". This image of a camel has become an unofficial symbol of Perl as well as a general hacker emblem, appearing on T-shirts and other clothing items.

O'Reilly owns the image as a trademark but licenses it for non-commercial use, requiring only an acknowledgement and a link to www.perl.com. Licensing for commercial use is decided on a case by case basis. O'Reilly also provides "Programming Republic of Perl" logos for non-commercial sites and "Powered by Perl" buttons for any site that uses Perl.



### Onion symbol

The Perl Foundation owns an alternative symbol, an onion, which it licenses to its subsidiaries, Perl Mongers, PerlMonks, Perl.org, and others. The symbol is a visual pun on pearl onion.



**Overview**

### Features

The overall structure of Perl derives broadly from C. Perl is procedural in nature, with variables, expressions, assignment statements, brace-delimited blocks, control structures, and subroutines.

Perl also takes features from shell programming. All variables are marked with leading sigils, which allow variables to be interpolated directly into strings. However, unlike the shell, Perl uses sigils on all accesses to variables, and unlike most other programming languages that use sigils, the sigil doesn't denote the type of the variable but the type of the expression. So for example, to access a list of values in a hash, the sigil for an array ("@") is used, not the sigil for a hash ("%"). Perl also has many built-in functions that provide tools often used in shell programming (although many of these tools are implemented by programs external to the shell) such as sorting, and calling operating system facilities.

Perl takes lists from Lisp, hashes ("associative arrays") from AWK, and regular expressions from sed. These simplify and facilitate many parsing, text-handling, and data-management tasks. Also shared with Lisp are the implicit return of the last value in a block, and the fact that all statements have a value, and thus are also expressions and can be used in larger expressions themselves.

Perl 5 added features that support complex data structures, first-class functions (that is, closures as values), and an object-oriented programming model. These include references, packages, class-based method dispatch, and lexically scoped variables, along with compiler directives (for example, the strict pragma). A major additional feature introduced with Perl 5 was the ability to package code as reusable modules. Wall later stated that "The whole intent of Perl 5's module system was to encourage the growth of Perl culture rather than the Perl core."[[65]](https://en.wikipedia.org/wiki/Perl#cite_note-65)

All versions of Perl do automatic data-typing and automatic memory management. The interpreter knows the type and storage requirements of every data object in the program; it allocates and frees storage for them as necessary using reference counting (so it cannot deallocate circular data structures without manual intervention). Legal type conversions — for example, conversions from number to string — are done automatically at run time; illegal type conversions are fatal errors.

### Design

The design of Perl can be understood as a response to three broad trends in the computer industry: falling hardware costs, rising labor costs, and improvements in compiler technology. Many earlier computer languages, such as Fortran and C, aimed to make efficient use of expensive computer hardware. In contrast, Perl was designed so that computer programmers could write programs more quickly and easily.

Perl has many features that ease the task of the programmer at the expense of greater CPU and memory requirements. These include automatic memory management; dynamic typing; strings, lists, and hashes; regular expressions; introspection; and an eval() function. Perl follows the theory of "no built-in limits", an idea similar to the Zero One Infinity rule.

Wall was trained as a linguist, and the design of Perl is very much informed by linguistic principles. Examples include Huffman coding (common constructions should be short), good end-weighting (the important information should come first), and a large collection of language primitives. Perl favors language constructs that are concise and natural for humans to write, even where they complicate the Perl interpreter.

Perl's syntax reflects the idea that "things that are different should look different." For example, scalars, arrays, and hashes have different leading sigils. Array indices and hash keys use different kinds of braces. Strings and regular expressions have different standard delimiters. This approach can be contrasted with a language such as Lisp, where the same basic syntax, composed of simple and universal symbolic expressions, is used for all purposes.

Perl does not enforce any particular programming paradigm (procedural, object-oriented, functional, or others) or even require the programmer to choose among them.

There is a broad practical bent to both the Perl language and the community and culture that surround it. The preface to *Programming Perl* begins: "Perl is a language for getting your job done." One consequence of this is that Perl is not a tidy language. It includes many features, tolerates exceptions to its rules, and employs heuristics to resolve syntactical ambiguities. Because of the forgiving nature of the compiler, bugs can sometimes be hard to find. Perl's function documentation remarks on the variant behavior of built-in functions in list and scalar contexts by saying, "In general, they do what you want, unless you want consistency."

No written specification or standard for the Perl language exists for Perl versions through Perl 5, and there are no plans to create one for the current version of Perl. There has been only one implementation of the interpreter, and the language has evolved along with it. That interpreter, together with its functional tests, stands as a *de facto* specification of the language. Perl 6, however, started with a specification, and several projects aim to implement some or all of the specification.

### Applications

Perl has many and varied applications, compounded by the availability of many standard and third-party modules.

Perl has chiefly been used to write CGI scripts: large projects written in Perl include cPanel, Slash, Bugzilla, RT, TWiki, and Movable Type; high-traffic websites that use Perl extensively include Priceline.com, Craigslist, IMDb, LiveJournal, DuckDuckGo, Slashdot and Ticketmaster. It is also an optional component of the popular LAMP technology stack for Web development, in lieu of PHP or Python.

Perl is often used as a glue language, tying together systems and interfaces that were not specifically designed to interoperate, and for "data munging",[[74]](https://en.wikipedia.org/wiki/Perl" \l "cite_note-74) that is, converting or processing large amounts of data for tasks such as creating reports. In fact, these strengths are intimately linked. The combination makes Perl a popular all-purpose language for system administrators, particularly because short programs, often called "one-liner programs", can be entered and run on a single command line.

Perl code can be made portable across Windows and Unix; such code is often used by suppliers of software (both COTS and bespoke) to simplify packaging and maintenance of software build- and deployment-scripts.

Graphical user interfaces (GUIs) may be developed using Perl. For example, Perl/Tk and wxPerl are commonly used to enable user interaction with Perl scripts. Such interaction may be synchronous or asynchronous, using callbacks to update the GUI.

### Implementation

Perl is implemented as a core interpreter, written in C, together with a large collection of modules, written in Perl and C. As of 2010, the interpreter is 150,000 lines of C code and compiles to a 1 MB executable on typical machine architectures. Alternatively, the interpreter can be compiled to a link library and embedded in other programs. There are nearly 500 modules in the distribution, comprising 200,000 lines of Perl and an additional 350,000 lines of C code (much of the C code in the modules consists of character encoding tables).

The interpreter has an object-oriented architecture. All of the elements of the Perl language—scalars, arrays, hashes, coderefs, file handles—are represented in the interpreter by C structs. Operations on these structs are defined by a large collection of macros, typedefs, and functions; these constitute the Perl C API. The Perl API can be bewildering to the uninitiated, but its entry points follow a consistent naming scheme, which provides guidance to those who use it.

The life of a Perl interpreter divides broadly into a compile phase and a run phase. In Perl, the **phases** are the major stages in the interpreter's life-cycle. Each interpreter goes through each phase only once, and the phases follow in a fixed sequence.

Most of what happens in Perl's compile phase is compilation, and most of what happens in Perl's run phase is execution, but there are significant exceptions. Perl makes important use of its capability to execute Perl code during the compile phase. Perl will also delay compilation into the run phase. The terms that indicate the kind of processing that is actually occurring at any moment are **compile time** and **run time**. Perl is in compile time at most points during the compile phase, but compile time may also be entered during the run phase. The compile time for code in a string argument passed to the eval built-in occurs during the run phase. Perl is often in run time during the compile phase and spends most of the run phase in run time. Code in BEGIN blocks executes at run time but in the compile phase.

At compile time, the interpreter parses Perl code into a syntax tree. At run time, it executes the program by walking the tree. Text is parsed only once, and the syntax tree is subject to optimization before it is executed, so that execution is relatively efficient. Compile-time optimizations on the syntax tree include constant folding and context propagation, but peephole optimization is also performed.

Perl has a Turing-complete grammar because parsing can be affected by run-time code executed during the compile phase. Therefore, Perl cannot be parsed by a straight Lex/Yacc lexer/parser combination. Instead, the interpreter implements its own lexer, which coordinates with a modified GNU bison parser to resolve ambiguities in the language.

It is often said that "Only perl can parse Perl", meaning that only the Perl interpreter (*perl*) can parse the Perl language (*Perl*), but even this is not, in general, true. Because the Perl interpreter can simulate a Turing machine during its compile phase, it would need to decide the halting problem in order to complete parsing in every case. It is a long-standing result that the halting problem is undecidable, and therefore not even perl can always parse Perl. Perl makes the unusual choice of giving the user access to its full programming power in its own compile phase. The cost in terms of theoretical purity is high, but practical inconvenience seems to be rare.

Other programs that undertake to parse Perl, such as source-code analyzers and auto-indenters, have to contend not only with ambiguous syntactic constructs but also with the undecidability of Perl parsing in the general case. Adam Kennedy's PPI project focused on parsing Perl code as a document (retaining its integrity as a document), instead of parsing Perl as executable code (that not even Perl itself can always do). It was Kennedy who first conjectured that "parsing Perl suffers from the 'halting problem'",which was later proved.

Perl is distributed with over 250,000 functional tests for core Perl language and over 250,000 functional tests for core modules. These run as part of the normal build process and extensively exercise the interpreter and its core modules. Perl developers rely on the functional tests to ensure that changes to the interpreter do not introduce software bugs; additionally, Perl users who see that the interpreter passes its functional tests on their system can have a high degree of confidence that it is working properly.

**Example code**

In older versions of Perl, one would write the Hello World program as:

**print "Hello, World!\n";**

Here is a more complex Perl program, that counts down the seconds up to a given threshold:

**#!/usr/bin/perl**

**use strict;**

**use warnings;**

**use IO::Handle;**

**my ( $remaining, $total );**

**$remaining = $total = shift(@ARGV);**

**STDOUT->autoflush(1);**

**while ( $remaining ) {**

**printf ( "Remaining %s/%s \r", $remaining--, $total );**

**sleep 1;**

**}**

**print "\n";**

The perl interpreter can also be used for one-off scripts on the command line. The following example (as invoked from an sh-compatible shell, such as Bash) translates the string "Bob" in all files ending with .txt in the current directory to "Robert":

**$ perl -i.bak -lp -e 's/Bob/Robert/g' \*.txt**