

The
Complete
Reference



Part I

Fundamentals

The
Complete
Reference



Chapter 1

Perl Background

Perl is many different things to many different people. The most fundamental aspect of Perl is that it's a high-level programming language written originally by Larry Wall and now supported and developed by a cast of thousands. The Perl language semantics are largely based on the C programming language, while also inheriting many of the best features of **sed**, **awk**, the Unix shell, and at least a dozen other tools and languages.

Although it is a bad idea to pigeonhole any language and assign it to a specific list of tasks, Perl is particularly strong at process, file, and text manipulation. This makes it especially useful for system utilities, software tools, systems management tasks, database access, graphical programming, networking, and web programming. These strengths make it particularly attractive to CGI script authors, systems administrators, mathematicians, journalists, and just about anybody who needs to write applications and utilities very quickly.

Perl has its roots firmly planted in the Unix environment, but it has since become a cross-platform development tool. Perl runs on IBM mainframes; AS/400s; Windows NT, 95, and 98; OS/2; Novell Netware; Cray supercomputers; Digital's VMS; Tandem Guardian; HP MPE/ix; Mac OS; and all flavors of Unix, including Linux. In addition, Perl has been ported to dozens of smaller operating systems, including BeOS, Acorn's RISCOS, and even machines such as the Amiga.

Larry Wall is a strong proponent of free software, and Perl is no exception. Perl, including the source code, the standard Perl library, the optional modules, and all of the documentation, is provided free and is supported entirely by its user community.

Before we get into the details of how to program in Perl, it's worth taking the time to familiarize yourself with where Perl has come from, what it can be used for, and how it stacks up against other languages. We'll also look at some popular "mythconceptions" about what Perl is and at some success stories of how Perl has helped a variety of organizations solve an equally varied range of problems.

What Does PERL Stand For?

There is a lot of controversy and rumor about exactly what PERL stands for and if, in fact, it stands for anything. According to Larry Wall, the original acronym stood for Practical Extraction and Reporting Language, and this relates to the original development purpose, which was to process a large amount of textual report information.

Over the years, other solutions have been proposed for the PERL acronym. The most popular recent version is Pathologically Eclectic Rubbish Lister. Luckily, a rough translation of that expansion equates to the original version!

Versions and Naming Conventions

The current version of Perl (at the time of writing—Nov 2000) was Perl 5.6, with a develop version, v5.7, already in production. Some sites are migrating to v5.6, others seem to be dragging their heels, although there are no major compatibility problems.

Up until March 2000, the situation concerning the available versions of Perl was quite complex, but we'll start with the "current" version first. From the release of Perl 5.6 there are two very simple strands. Even version numbers, such as 5.6 and 5.8 are considered to be "stable" releases of the language. Odd version numbers, such as 5.7 and 5.9, are development releases.

Perl 5.6 was a long time coming—over two years since the last major release—but it also set a landmark for Perl's development. It was the first version that really reunited the core and Win32 versions of Perl, as well as providing some compatibility enhancements. For example, the Windows ports now support **fork**, something not natively provided by the Windows operating system. Also updated were the Perl compiler and the threading system (which actually supports the Windows **fork** function), and the addition of a new keyword, **our**, which handles global variables in the same way as **my**.

Discussions have already started for Perl 6. Unlike Perl 5, which was a complete rewrite of Perl 4 and was developed and coded almost entirely by Larry, Perl 6 will have its feature set determined by the people that use it, through a series of RFCs (Requests for Comments). The language's core code will be developed by a team of programmers with input and assistance from Larry, and with features agreed upon by committees, rather than solely by Larry. This will make Perl 6 a language designed by the people that use it, rather than by the person who invented it.

Perl, perl or PeRI?

There is also a certain amount of confusion regarding the capitalization of Perl. Should it be written Perl or perl? Larry Wall now uses "Perl" to signify the language proper and "perl" to signify the implementation of the language. Therefore, perl can parse Perl. In essence, however, it really doesn't make a huge amount of difference. That said, you will find that the executable version of perl is installed with its name in lowercase!

Life Before Perl 5.6

Before Perl 5.6, version numbers were far more confusing. Before version 5 came version 4, the highest incarnation of which was 4.036, released in 1993. Version 5 is still in development, with version 5.005_03 being the last stable release before the

current 5.6. However, many sites were using Perl 5.005_56—this was a developmental release, but stable enough that some sites used it in preference to 5.005_02. Although there were changes between these versions, they were bug fixes rather than the significant improvements in Perl 5.6.

As to naming, you will see references to perl4 and perl5, and more recently, perl5.6. Since most people will be using at least perl5, it's probably safe to refer to Perl simply as Perl!

Perl History

Perl is a relatively old language, with the first version having been released in 1988. The basic history is shown in Table 1-1.

If you want a more detailed history of Perl, check out the **perlhists** documentation installed with Perl, or visit CFAST, the Comprehensive Perl Arcana Society Tapestry at history.perl.org.

Version	Date	Version Details
Perl 0		Introduced Perl to Larry Wall's office associates
Perl 1	Jan 1988	Introduced Perl to the world
Perl 2	Jun 1988	Introduced Harry Spencer's regular expression package
Perl 3	Oct 1989	Introduced the ability to handle binary data
Perl 4	Mar 1991	Introduced the first "Camel" book (<i>Programming Perl</i> , by Larry Wall, Tom Christiansen, and Randal L Schwartz; O'Reilly & Associates). The book drove the name change, just so it could refer to Perl 4, instead of Perl 3.
Perl 4.036	Feb 1993	The last stable release of Perl 4
Perl 5	Oct 1994	The first stable release of Perl 5, which introduced a number of new features and a complete rewrite.
Perl 5.005_02	Aug 1998	The next major stable release
Perl 5.005_03	Mar 1999	The last stable release before 5.6
Perl 5.6	Mar 2000	Introduced unified fork support, better threading, an updated Perl compiler, and the our keyword

Table 1-1. *Perl Version History*

Main Perl Features

Perl contains many features that most Perl programmers do not even know about, let alone use. Some of the most basic features are described here.

Perl Is Free

It may not seem like a major feature, but, in fact, being free is very important. Some languages, such as C (which is free with compilers such as GNU's `gcc`), have been commercialized by Metrowerks, Microsoft, and other companies. Other languages, such as Visual Basic, are entirely commercial. Perl's source code is open and free—anybody can download the C source that constitutes a Perl interpreter. Furthermore, you can easily extend the core functionality of Perl both within the realms of the interpreted language and by modifying the Perl source code.

Perl Is Simple to Learn, Concise, and Easy to Read

Because of its history and roots, most people with any programming experience will be able to program with Perl. It has a syntax similar to C and shell script, among others, but with a less restrictive format. Most programs are quicker to write in Perl because of its use of built-in functions and a huge standard and contributed library. Most programs are also quicker to execute than other languages because of Perl's internal architecture (see the section, "Perl is Fast" that follows). Perl can be easy to read, because the code can be written in a clear and concise format that almost reads like an English sentence. Unfortunately, Perl also has a bad habit of looking a bit like line noise to uninitiated. Whether or not your Perl looks good and clean really depends on how you format it—good Perl is easy read. It is also worth reading the Perl style guidelines (in the Perl style manual page that comes with Perl) to see how Larry Wall, Perl's creator, likes things done.

Perl Is Fast

As we will see shortly, Perl is not an interpreter in the strictest sense—when you execute a Perl program it is actually compiled into a highly optimized language before it is executed. Compared to most scripting languages, this makes execution almost as fast as compiled C code. But, because the code is still interpreted, there is no compilation process, and applications can be written and edited much faster than with other languages, without any of the performance problems normally associated with an interpreted language.

Perl Is Extensible

You can write Perl-based packages and modules that extend the functionality of the language. You can also call external C code directly from Perl to extend the functionality

further. The reverse is also true: the Perl interpreter can be incorporated directly into many languages, including C. This allows your C programs to use the functionality of the Perl interpreter without calling an external program.

Perl Has Flexible Data Types

You can create simple variables that contain text or numbers, and Perl will treat the variable data accordingly at the time it is used. This means that unlike C, you don't have to worry about converting text and numbers, and you can embed and merge strings without requiring external functions to concatenate or combine the results. You can also handle arrays of values as simple lists, as typical indexed arrays, and even as stacks of information. You can also create associative arrays (otherwise known as hashes) which allow you to refer to the items in the array by a unique string, rather than a simple number. Finally, Perl also supports references, and through references objects. References allow you to create complex data structures made up of a combination of hashes, lists and scalars.

Perl Is Object Oriented

Perl supports all of the object-oriented features—inheritance, polymorphism, and encapsulation. There are no restrictions on when or where you make use of object-oriented features. There is no boundary as there is with C and C++.

Perl Is Collaborative

There is a huge network of Perl programmers worldwide. Most programmers supply, and use, the modules and scripts available via CPAN, the Comprehensive Perl Archive Network (see Web Appendix B at www.osborne.com). This is a repository of the best modules and scripts available. Using an existing prewritten module can save you hundreds, perhaps even thousands, of hours of development time.

Compiler or Interpreter

Different languages work in different ways; they are either compiled or interpreted. A program in a compiled language is translated from the original source into a platform-specific machine code. This machine code is referred to as an *executable*. There is no direct relation between the machine code and the original source: it is not possible to reverse the compilation process and produce the source code. This means that the compiled executable is safe from intellectual property piracy.

With an interpreted language, on the other hand, the interpreter reads the original source code and interprets each of the statements in order to perform the different operations. The source code is therefore executed at run time. This has some advantages: Because there is no compilation process, the development of interpreted code should

be significantly quicker. Interpreted code also tends to be smaller and easier to distribute. The disadvantages are that the original source must be supplied in order to execute the program, and an interpreted program is generally slower than a compiled executable because of the way the code is executed.

Perl fits neither of these descriptions in the real sense. The internals of Perl are such that at the time of executing a Perl script, the individual elements of the script are compiled into a tree of *opcodes*. Opcodes are similar in concept to machine code—the binary format required by the processor in your machine. However, whereas machine code is executed directly by hardware, opcodes are executed by a Perl virtual machine. The opcodes are highly optimized objects designed to perform a specific function. When the script is executed you are essentially executing compiled C code, translated from the Perl source. This enables Perl to provide all the advantages of a scripting language while offering the fast execution of a compiled program. This mode of operation—translation and then execution by a virtual machine is actually how most modern scripting languages work, including Java (using Just In Time technology) and Python.

Keeping all of that in mind, however, there have been some advances in the most recent versions of a Perl compiler that takes native Perl scripts and converts them into directly executable machine code. We'll cover the compiler and Perl internals later in this book.

Similar Programming Languages

We already know that Perl has its history in a number of different languages. It shares several features and abilities with many of the standard tools supplied with any Unix workstation. It also shares some features and abilities with many related languages, even if it doesn't necessarily share the same heritage.

With regard to specific features, abilities, and performance, Perl compares favorably against some languages and less favorably against others. A lot of the advantages and disadvantages are a matter of personal preference. For example, for text handling, there is very little to choose between **awk** and Perl. However, personally I prefer Perl for those tasks that involve file handling directly within the code, and **awk** when using it as a filter as part of a shell script.

Unix Shells

Any of the Unix shells—**sh**, **csh**, **ksh**, or even **bash**—share the same basic set of facilities. They are particularly good at running external programs and at most forms of file management where the shell's ability to work directly with many of the standard Unix utilities enables rapid development of systems management tools.

However, where most shells fail is in their variable- and data-handling routines. In nearly all cases you need to use the facilities provided by shell tools such as **cut**, **paste**, and **sort** to achieve the same level of functionality as that provided natively by Perl.

Tcl

Tcl (Tool Command Language) was developed as an embeddable scripting language. A lot of the original design centered around a macro-like language for helping with shell-based applications. Tcl was never really developed as a general-purpose scripting language, although many people use it as such. In fact, Tcl was designed with the philosophy that you should actually use two or more languages when developing large software systems.

Tcl's variables are very different from those in Perl. Because it was designed with the typical shell-based string handling in mind, strings are null terminated (as they are in C). This means that Tcl cannot be used for handling binary data. Compared to Perl, Tcl is also generally slower on iterative operations over strings. You cannot pass arrays by value or by reference; they can only be passed by name. This makes programming more complex, although not impossible.

Lists in Tcl are actually stored as a single string, and arrays are stored within what Perl would treat as a hash. Accessing a true Tcl array is therefore slightly slower, as it has to look up associative entries in order to decipher the true values. The data-handling problems also extend to numbers, which Tcl stores as strings and converts to numbers only when a calculation is required. This slows mathematical operations significantly.

Unlike Perl, which parses the script first before optimizing and then executing, Tcl is a true interpreter, and each line is interpreted and optimized individually at execution time. This reduces the optimization options available to Tcl. Perl, on the other hand, can optimize source lines, code blocks, and even entire functions if the compilation process allows. The same Tcl interpretation technique also means that the only way to debug Tcl code and search for syntactic errors is to actually execute the code. Because Perl goes through the precompilation stage, it can check for syntactic and other possible or probable errors without actually executing the code.

Finally, the code base of the standard Tcl package does not include many of the functions and abilities of the Perl language. This is especially important if you are trying to write a cross-platform POSIX-compliant application. Perl supports the entire POSIX function set, but Tcl supports a much smaller subset of the POSIX function set, even using external packages.

It should be clear from this description that Perl is a better alternative to Tcl in situations where you want easy access to the rest of the OS. Most significantly, Tcl will never be a general-purpose scripting language. Tcl will, on the other hand, be a good solution if you want to embed a scripting language inside another language.

Python

Python was developed as an object-oriented language and is well thought out. It is an interpreted, byte-compiled, extensible, and largely procedural programming language.

Like Perl, it's good at text processing and even general-purpose programming. Python also has a good history in the realm of GUI-based application development. Compared to Perl, Python has fewer users, but it is gaining acceptance as a practical rapid application development tool.

Unlike Perl, Python does not resemble C, and it doesn't resemble Unix-style tools like **awk** either. Python was designed from scratch to be object oriented and has clear module semantics. This can make it confusing to use, as the name spaces get complex to resolve. On the other hand, this makes it much more structured, which can ease development for those with structured minds.

I'm not aware of anything that is better in Python than in Perl. They both share object features, and the two are almost identical in execution speed. However, the reverse is not true: Perl has better regular expression features, and the level of integration between Perl and the Unix environment is hard to beat (although it can probably be solved within Python using a suitably written external module).

In general, there is not a lot to tip the scales in favor of one of the two languages. Perl will appeal to those people who already know C or Unix shell utilities. Perl is also older and more widespread, and there is a much larger library of contributed modules and scripts. Python, on the other hand, may appeal to those people who have experience with more object-oriented languages, such as Java or Modula-2.

Both languages provide easy control and access when it comes to the external environment in which they work. Perl arguably fills the role better, though, because many of the standard system functions you are used to are supported natively by the language, without requiring external modules. The technical support for the two languages is also very similar, with both using websites and newsgroups to help users program in the new language.

Finally, it's worth mentioning that of all the scripting languages available, Perl and Python are two of the most stable platforms for development. There are, however, some minor differences. First, Perl provides quite advanced functions and mechanisms for tracking errors and faults in the scripts. Making extensive use of these facilities can still cause problems, however. For example, calling the system **truncate()** function within Perl will cause the whole interpreter to crash. Python, on the other hand, uses a system of error trapping that will immediately identify a problem like this before it occurs, allowing you to account for it in your applications. This is largely due to the application-development nature of the language.

Java

At first viewing, Java seems to be a friendlier, interpreted version of C++. Depending on your point of view, this can either be an advantage or a disadvantage. Java probably inherits less than a third of the complexity of C++, but it retains much of the complexity of its brethren.

Java was designed primarily as an implementation-independent language, originally with web-based intentions, but now as a more general-purpose solution to a variety of problems. Like Perl, Java is byte compiled, but unlike Perl, programs are supplied in byte-compiled format and then executed via a Java virtual machine at execution time.

Because of its roots and its complexity, Java cannot really be considered as a direct competitor to Perl. It is difficult to use Java as a rapid application development tool and virtually impossible to use it for most of the simple text-processing and system-administration tasks that Perl is best known for.

C/C++

Perl itself is written in C. (You can download and view the Perl source code if you so wish, but it's not for the faint-hearted!) Many of the structures and semantics of Perl and C are very similar. For example, both use semicolons as end-of-line terminators. They also share the same code block and indentation features. However, Perl tends to be stricter when it comes to code block definitions—it always requires curly brackets, for example—but most C programmers will be comfortable with the Perl environment.

Perl can be object oriented like C++. Both share the same abilities of inheritance, polymorphism, and encapsulation. However, object orientation in Perl is easier to use, compared to the complexities of constructors and inheritance found in C++. In addition to all this, there is no distinction between the standard and object-oriented implementations of Perl as there is with C and C++. This means you can mix and match different variables, objects, and other data types within a single Perl application—something that would be difficult to achieve easily with C and C++.

Because Perl is basically an interpreted language (as mentioned earlier), development is generally quicker than is writing in native C. Perl also has many more built-in facilities and abilities that would otherwise need to be handwritten in C/C++. For example, regular expressions and many of the data-handling features would require a significant amount of programming to reproduce in C with the same ease of use available in Perl.

Because of Perl's roots in C, it is also possible to extend Perl with C source code and vice versa: you can embed Perl programs in C source code.

awk/gawk

Although a lot of syntax is different, **awk**, and **gawk** (the GNU projects version) are functionally subsets of Perl. It's also clear from the history of Perl that many of the features have been inherited directly from those of **awk**. Indeed, **awk** was designed as a reporting language with the emphasis on making the process of reporting via the shell significantly easier. Without **awk**, you would have to employ a number of external utilities, such as **cut**, **expr**, and **sort**, and the solution would be neither quick nor elegant.

There are some things that Perl has built-in support for that **awk** does not. For example, **awk** has no network socket class, and it is largely ignorant of external files,

when compared to the file manipulation and management functions found in Perl. However, some advantages **awk** has over Perl are summarized here:

- **awk** is simpler, and the syntax is more structured and regular.
- Although it is gaining acceptance, Perl has yet to be included as standard with many operating systems. **Awk** has been supplied with Unix almost since it was first released.
- **awk** can be smaller and therefore much quicker to execute for small programs.
- **awk** supports more advanced regular expressions. You can use a regular expression for replacement, and you can search text in substitutions.

Popular “Mythconceptions”

Despite its history and wide use in many different areas, there are still a number of myths about what Perl is, where it should be used, and even why it was invented. Here’s a quick list of the popular mythconceptions of the Perl language.

It’s Only for the Web

Probably the most famous of the myths is that Perl is a language used, designed, and created exclusively for developing web-based applications. In fact, this could not be more wrong. Version 1.0 of Perl, the first released to the world, shipped in 1988—several years before the web and HTML as we know it today were in general use. In fact, Perl was inherited as a good design tool for web server applications based on its ease of use and flexibility. The text-handling features are especially useful when working within the web environment. There are libraries of database interfaces, client-server modules, networking features, and even GUI toolkits to enable you to write entire applications directly within Perl.

It’s Not Maintenance Friendly

Any good (or bad) programmer will tell you that anybody can write unmaintainable code in any language. Many companies and individuals write maintainable programs using Perl. A lot of people would argue that Perl’s structured style, easily readable source code, and modular format make it more maintainable than languages such as C, C++, and Java.

It’s Only for Hackers

Perl is used by a variety of companies, organizations, and individuals. Everybody from programming beginners through “hackers” up to multinational corporations use Perl to solve their problems. It can hardly be classed as a hackers-only language. Moreover,

it is maintained by the same range of people, which means you get the best of both worlds—real-world features, with top-class behind-the-scenes algorithms.

It's a Scripting Language

In Perl, there is no difference between a script and program. Many large programs and projects have been written entirely in Perl. A good example is *Majordomo*, the main mailing-list manager used on the Internet. It's written entirely in Perl. See the upcoming section "Perl Success Stories" for more examples of where Perl has made a difference, despite its scripting label.

There's No Support

The Perl community is one of the largest on the Internet, and you should be able to find someone, somewhere, who can answer your questions or help you with your problems. The Perl Clinic (see Appendix C) offers free advice and support to Perl programmers.

All Perl Programs Are Free

Although you generally write and use Perl programs in their native source form, this does not mean that everything you write is free. Perl programs are your own intellectual property and can be bought, sold, and licensed just like any other program. If you are worried about somebody stealing your code, source filters and bytecode compilers will render your code useful only for execution and unreadable by the casual software pirate.

There's No Development Environment

Development environments are only really required when you need to compile source code into object files. Because Perl scripts are written in normal text, you can use any editor to write and use Perl programs. Under Unix, the favorites are **emacs** and **vi**, and both have Perl modes to make syntax checking and formatting easier. Under Windows NT, you can also use **emacs**, or you can use Solutionsoft's Perl Builder, which is an interactive environment for Perl programs. Alternatively, you can use the ActiveState debugger, which will provide you with a direct environment for executing and editing Perl statements. There are also many improvements being made in the ActiveState distribution that will allow Perl to be used as part of Microsoft's Visual Studio product under a project called VisualPerl. On the Mac, the BBEdit and Pepper editors have a Perl mode that colors the syntax of the Perl source to make it easier to read.

Additionally, because Perl programs are text based, you can use any source-code revision-control system. The most popular solution is CVS, or Concurrent Versioning System, which is now supported under Unix, MacOS and Windows.

Perl Is a GNU Project

While the GNU project includes Perl in its distributions, there is no such thing as “GNU Perl.” Perl is not produced or maintained by GNU and the Free Software Foundation. Perl is also made available on a much more open license than the GNU Public License.

Note

GNU stands for the recursive “GNU’s Not Unix,” and is part of the Free Software Foundation, an organization devoted to providing a suite of useful user software for free.

Perl Is Difficult to Learn

Because Perl is similar to a number of different languages, it is not only easy to learn but also easy to continue learning. Its structure and format is very similar to C, **awk**, shell script, and, to a greater or lesser extent, even BASIC. If you have ever done any form of programming, you’re half way toward learning programming in Perl.

In many cases, you will only use a very small subset of Perl to complete most tasks. The guiding motto for Perl development is “there’s more than one way to do it.” This makes Perl’s learning curve very shallow and very long. Perl is a large language with a great many features, and there is a lot you can learn if you want to.

Perl Success Stories

Perl has been used by thousands of different corporations to tackle and solve different problems. For most people, it has reduced the development time for their desired application by days, weeks, or even months. Below is a sample of the bigger companies that have used Perl. I’ve tried to include testimonials and deeper examples of how Perl was the better solution, where the information has been available.

- **Amazon.com**, one of the Internet’s best known and most successful e-commerce sites, used Perl to develop an entire editorial production and control system. This integrates the authoring, maintenance (including version control and searching), and output of the editorial content of the entire Amazon.com website.
- Netscape engineers wrote a content management and delivery system, with logging, analysis, and feedback on use, in three months using Perl.
- In order to get around many cross-platform development problems, SPEC (the Standard Performance Evaluation Corporation) used Perl as a wrapper around the C code that is used to test performance. With Perl’s ability to import and dynamically use external C code in combination with its object-oriented abilities, SPEC generated a test system that was easily portable from Unix to the Windows NT platform.

- Using an old 60MHz Pentium and Perl, a New England hospital implemented a distributed printing system that connected 20,000 PC workstations to 3,000 printers spread over an entire city.

On a personal level, I have Perl scripts that create users, add new virtual WWW servers to Apache, monitor all the machines and storage on my network, keep track of all my archives and e-mail, and even scripts that download weather information from my weather center and get the TV listings every day!