NAME

perlfaq1 - General Questions About Perl

DESCRIPTION

This section of the FAQ answers very general, high-level questions about Perl.

What is Perl?

Perl is a high-level programming language with an eclectic heritage written by Larry Wall and a cast of thousands. It derives from the ubiquitous C programming language and to a lesser extent from sed, awk, the Unix shell, and at least a dozen other tools and languages. Perl's process, file, and text manipulation facilities make it particularly well-suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking, and world wide web programming. These strengths make it especially popular with system administrators and CGI script authors, but mathematicians, geneticists, journalists, and even managers also use Perl. Maybe you should, too.

Who supports Perl? Who develops it? Why is it free?

The original culture of the pre-populist Internet and the deeply-held beliefs of Perl's author, Larry Wall, gave rise to the free and open distribution policy of perl. Perl is supported by its users. The core, the standard Perl library, the optional modules, and the documentation you're reading now were all written by volunteers. See the personal note at the end of the README file in the perl source distribution for more details. See *perlhist* (new as of 5.005) for Perl's milestone releases.

In particular, the core development team (known as the Perl Porters) are a rag-tag band of highly altruistic individuals committed to producing better software for free than you could hope to purchase for money. You may snoop on pending developments via the archives at http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/ and http://archive.develooper.com/perl5-porters@perl.org/ or the news gateway nntp://nntp.perl.org/perl.perl5.porters or its web interface at http://nntp.perl.org/group/perl.perl5.porters , or read the faq at http://dev.perl.org/perl5/docs/p5p-faq.html , or you can subscribe to the mailing list by sending perl5-porters-subscribe@perl.org a subscription request (an empty message with no subject is fine).

While the GNU project includes Perl in its distributions, there's no such thing as "GNU Perl". Perl is not produced nor maintained by the Free Software Foundation. Perl's licensing terms are also more open than GNU software's tend to be.

You can get commercial support of Perl if you wish, although for most users the informal support will more than suffice. See the answer to "Where can I buy a commercial version of perl?" for more information.

Which version of Perl should I use?

(contributed by brian d foy)

There is often a matter of opinion and taste, and there isn't any one answer that fits everyone. In general, you want to use either the current stable release, or the stable release immediately prior to that one. Currently, those are perl5.14.x and perl5.12.x, respectively.

Beyond that, you have to consider several things and decide which is best for you.

- If things aren't broken, upgrading perl may break them (or at least issue new warnings).
- The latest versions of perl have more bug fixes.
- The Perl community is geared toward supporting the most recent releases, so you'll have an easier time finding help for those.
- Versions prior to perl5.004 had serious security problems with buffer overflows, and in some cases have CERT advisories (for instance, http://www.cert.org/advisories/CA-1997-17.html).

- The latest versions are probably the least deployed and widely tested, so you may want to wait a few months after their release and see what problems others have if you are risk averse.
- The immediate, previous releases (i.e. perl5.8.x) are usually maintained for a while, although not at the same level as the current releases.
- No one is actively supporting Perl 4. Ten years ago it was a dead camel carcass (according to this document). Now it's barely a skeleton as its whitewashed bones have fractured or eroded.
- There is no Perl 6 release scheduled, but it will be available when it's ready. The joke is that it's scheduled for Christmas, but that we just don't know which one. Stay tuned, but don't worry that you'll have to change major versions of Perl; no one is going to take Perl 5 away from you.
- There are really two tracks of perl development: a maintenance version and an experimental version. The maintenance versions are stable, and have an even number as the minor release (i.e. perl5.10.x, where 10 is the minor release). The experimental versions may include features that don't make it into the stable versions, and have an odd number as the minor release (i.e. perl5.9.x, where 9 is the minor release).

What are Perl 4, Perl 5, or Perl 6?

(contributed by brian d foy)

In short, Perl 4 is the past, Perl 5 is the present, and Perl 6 is the future.

The number after Perl (i.e. the 5 after Perl 5) is the major release of the perl interpreter as well as the version of the language. Each major version has significant differences that earlier versions cannot support.

The current major release of Perl is Perl 5, and was first released in 1994. It can run scripts from the previous major release, Perl 4 (March 1991), but has significant differences. It introduced the concept of references, complex data structures, and modules. The Perl 5 interpreter was a complete re-write of the previous perl sources.

Perl 6 is the next major version of Perl, although it's not intended to replace Perl 5. It's still in development in both its syntax and design. The work started in 2002 and is still ongoing. Some of the most interesting features have shown up in the latest versions of Perl 5, and some Perl 5 modules allow you to use some Perl 6 syntax in your programs. The current leading implementation of Perl 6 is Rakudo (http://rakudo.org).

See *perlhist* for a history of Perl revisions.

What was Ponie?

(contributed by brian d foy)

Ponie stands for "Perl On the New Internal Engine", started by Arthur Bergman from Fotango in 2003, and subsequently run as a project of The Perl Foundation. It was abandoned in 2006 (http://www.nntp.perl.org/group/perl.ponie.dev/487).

Instead of using the current Perl internals, Ponie aimed to create a new one that would provide a translation path from Perl 5 to Perl 6 (or anything else that targets Parrot, actually). You would have been able to just keep using Perl 5 with Parrot, the virtual machine which will compile and run Perl 6 bytecode.

What is Perl 6?

At The Second O'Reilly Open Source Software Convention, Larry Wall announced Perl 6 development would begin in earnest. Perl 6 was an oft used term for Chip Salzenberg's project to rewrite Perl in C++ named Topaz. However, Topaz provided valuable insights to the next version of Perl and its implementation, but was ultimately abandoned.

If you want to learn more about Perl 6, or have a desire to help in the crusade to make Perl a better place then read the Perl 6 developers page at http://dev.perl.org/perl6/ and get involved.

Perl 6 is not scheduled for release yet, and Perl 5 will still be supported for quite awhile after its release. Do not wait for Perl 6 to do whatever you need to do.

"We're really serious about reinventing everything that needs reinventing." --Larry Wall

How stable is Perl?

Production releases, which incorporate bug fixes and new functionality, are widely tested before release. Since the 5.000 release, we have averaged only about one production release per year.

Larry and the Perl development team occasionally make changes to the internal core of the language, but all possible efforts are made toward backward compatibility. While not quite all Perl 4 scripts run flawlessly under Perl 5, an update to perl should nearly never invalidate a program written for an earlier version of perl (barring accidental bug fixes and the rare new keyword).

Is Perl difficult to learn?

No, Perl is easy to start learning--and easy to keep learning. It looks like most programming languages you're likely to have experience with, so if you've ever written a C program, an awk script, a shell script, or even a BASIC program, you're already partway there.

Most tasks only require a small subset of the Perl language. One of the guiding mottos for Perl development is "there's more than one way to do it" (TMTOWTDI, sometimes pronounced "tim toady"). Perl's learning curve is therefore shallow (easy to learn) and long (there's a whole lot you can do if you really want).

Finally, because Perl is frequently (but not always, and certainly not by definition) an interpreted language, you can write your programs and test them without an intermediate compilation step, allowing you to experiment and test/debug quickly and easily. This ease of experimentation flattens the learning curve even more.

Things that make Perl easier to learn: Unix experience, almost any kind of programming experience, an understanding of regular expressions, and the ability to understand other people's code. If there's something you need to do, then it's probably already been done, and a working example is usually available for free. Don't forget Perl modules, either. They're discussed in Part 3 of this FAQ, along with CPAN, which is discussed in Part 2.

How does Perl compare with other languages like Java, Python, REXX, Scheme, or Tcl?

Favorably in some areas, unfavorably in others. Precisely which areas are good and bad is often a personal choice, so asking this question on Usenet runs a strong risk of starting an unproductive Holy War.

Probably the best thing to do is try to write equivalent code to do a set of tasks. These languages have their own newsgroups in which you can learn about (but hopefully not argue about) them.

Some comparison documents can be found at http://www.perl.com/doc/FMTEYEWTK/versus/ if you really can't stop yourself.

Can I do [task] in Perl?

Perl is flexible and extensible enough for you to use on virtually any task, from one-line file-processing tasks to large, elaborate systems. For many people, Perl serves as a great replacement for shell scripting. For others, it serves as a convenient, high-level replacement for most of what they'd program in low-level languages like C or C++. It's ultimately up to you (and possibly your management) which tasks you'll use Perl for and which you won't.

If you have a library that provides an API, you can make any component of it available as just another Perl function or variable using a Perl extension written in C or C++ and dynamically linked into your main perl interpreter. You can also go the other direction, and write your main program in C or C++,

and then link in some Perl code on the fly, to create a powerful application. See *perlembed*.

That said, there will always be small, focused, special-purpose languages dedicated to a specific problem domain that are simply more convenient for certain kinds of problems. Perl tries to be all things to all people, but nothing special to anyone. Examples of specialized languages that come to mind include prolog and matlab.

When shouldn't I program in Perl?

When your manager forbids it--but do consider replacing them :-).

Actually, one good reason is when you already have an existing application written in another language that's all done (and done well), or you have an application language specifically designed for a certain task (e.g. prolog, make).

For various reasons, Perl is probably not well-suited for real-time embedded systems, low-level operating systems development work like device drivers or context-switching code, complex multi-threaded shared-memory applications, or extremely large applications. You'll notice that perl is not itself written in Perl.

Perl remains fundamentally a dynamically typed language, not a statically typed one. You certainly won't be chastised if you don't trust nuclear-plant or brain-surgery monitoring code to it. And Larry will sleep easier, too--Wall Street programs not withstanding. :-)

What's the difference between "perl" and "Perl"?

One bit. Oh, you weren't talking ASCII? :-) Larry now uses "Perl" to signify the language proper and "perl" the implementation of it, i.e. the current interpreter. Hence Tom's quip that "Nothing but perl can parse Perl."

Before the first edition of *Programming perl*, people commonly referred to the language as "perl", and its name appeared that way in the title because it referred to the interpreter. In the book, Randal Schwartz capitalised the language's name to make it stand out better when typeset. This convention was adopted by the community, and the second edition became *Programming Perl*, using the capitalized version of the name to refer to the language.

You may or may not choose to follow this usage. For example, parallelism means "awk and perl" and "Python and Perl" look good, while "awk and Perl" and "Python and perl" do not. But never write "PERL", because perl is not an acronym, apocryphal folklore and post-facto expansions notwithstanding.

Is it a Perl program or a Perl script?

Larry doesn't really care. He says (half in jest) that "a script is what you give the actors. A program is what you give the audience."

Originally, a script was a canned sequence of normally interactive commands--that is, a chat script. Something like a UUCP or PPP chat script or an expect script fits the bill nicely, as do configuration scripts run by a program at its start up, such *.cshrc* or *.ircrc*, for example. Chat scripts were just drivers for existing programs, not stand-alone programs in their own right.

A computer scientist will correctly explain that all programs are interpreted and that the only question is at what level. But if you ask this question of someone who isn't a computer scientist, they might tell you that a *program* has been compiled to physical machine code once and can then be run multiple times, whereas a *script* must be translated by a program each time it's used.

Now that "script" and "scripting" are terms that have been seized by unscrupulous or unknowing marketeers for their own nefarious purposes, they have begun to take on strange and often pejorative meanings, like "non serious" or "not real programming". Consequently, some Perl programmers prefer to avoid them altogether.

What is a JAPH?

(contributed by brian d foy)

JAPH stands for "Just another Perl hacker,", which Randal Schwartz used to sign email and usenet messages starting in the late 1980s. He previously used the phrase with many subjects ("Just another x hacker,"), so to distinguish his JAPH, he started to write them as Perl programs:

```
print "Just another Perl hacker,";
```

Other people picked up on this and started to write clever or obfuscated programs to produce the same output, spinning things quickly out of control while still providing hours of amusement for their creators and readers.

CPAN has several JAPH programs at http://www.cpan.org/misc/japh .

Where can I get a list of Larry Wall witticisms?

(contributed by brian d foy)

Google "larry wall quotes"! You might even try the "I feel lucky" button. :)

Wikiquote has the witticisms from Larry along with their source, including his usenet postings and source code comments.

If you want a plain text file, try http://www.cpan.org/misc/lwall-quotes.txt.gz .

How can I convince others to use Perl?

(contributed by brian d foy)

Appeal to their self interest! If Perl is new (and thus scary) to them, find something that Perl can do to solve one of their problems. That might mean that Perl either saves them something (time, headaches, money) or gives them something (flexibility, power, testability).

In general, the benefit of a language is closely related to the skill of the people using that language. If you or your team can be faster, better, and stronger through Perl, you'll deliver more value. Remember, people often respond better to what they get out of it. If you run into resistance, figure out what those people get out of the other choice and how Perl might satisfy that requirement.

You don't have to worry about finding or paying for Perl; it's freely available and several popular operating systems come with Perl. Community support in places such as Perlmonks (http://www.perlmonks.com) and the various Perl mailing lists (http://lists.perl.org) means that you can usually get quick answers to your problems.

Finally, keep in mind that Perl might not be the right tool for every job. You're a much better advocate if your claims are reasonable and grounded in reality. Dogmatically advocating anything tends to make people discount your message. Be honest about possible disadvantages to your choice of Perl since any choice has trade-offs.

You might find these links useful:

* http://perltraining.com.au/whyperl.html

* http://www.perl.org/advocacy/whyperl.html

AUTHOR AND COPYRIGHT

you see fit. A simple comment in the code giving credit to the FAQ would be courteous but is not required.