NAME

perlfaq1 - General Questions About Perl

VERSION

version 5.021009

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.

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 web programming.

Perl derives from the ubiquitous C programming language and to a lesser extent from sed, awk, the Unix shell, and many other tools and languages.

These strengths make it especially popular with web developers and system administrators. Mathematicians, geneticists, journalists, managers and many other people also use Perl.

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.

The core development team (known as the Perl Porters) are a group 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* or read the *faq*, 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 perl5.18.x and perl5.16.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. perl5.14.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.
- The current leading implementation of Perl 6, Rakudo, released a "useful, usable, 'early adopter'" distribution of Perl 6 (called Rakudo Star) in July of 2010. Please see *http://rakudo.org/* for more information.
- 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. perl5.18.x, where 18 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. perl5.19.x, where 19 is the minor release).

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

In short, Perl 4 is the parent to both Perl 5 and Perl 6. Perl 5 is the older sibling, and though they are different languages, someone who knows one will spot many similarities in the other.

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, first released in 1994. It can run scripts from the previous major release, Perl 4 (March 1991), but has significant differences.

Perl 6 is a reinvention of Perl, it is a language in the same lineage but not compatible. The two are complementary, not mutually exclusive. Perl 6 is not meant to replace Perl 5, and vice versa. See *What is Perl 6?* below to find out more.

See *perlhist* for a history of Perl revisions.

What is Perl 6?

Perl 6 was *originally* described as the community's rewrite of Perl 5. Development started in 2002; syntax and design work continue to this day. As the language has evolved, it has become clear that it is a separate language, incompatible with Perl 5 but in the same language family.

Contrary to popular belief, Perl 6 and Perl 5 peacefully coexist with one another. Perl 6 has proven to be a fascinating source of ideas for those using Perl 5 (the *Moose* object system is a well-known example). There is overlap in the communities, and this overlap fosters the tradition of sharing and borrowing that have been instrumental to Perl's success. The current leading implementation of Perl 6 is Rakudo, and you can learn more about it at *http://rakudo.org*.

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://www.perl6.org/* and get involved.

"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 about one production release per year.

The Perl development team occasionally make changes to the internal core of the language, but all

possible efforts are made toward backward compatibility.

How often are new versions of Perl released?

Recently, the plan has been to release a new version of Perl roughly every April, but getting the release right is more important than sticking rigidly to a calendar date, so the release date is somewhat flexible. The historical release dates can be viewed at *http://www.cpan.org/src/README.html*.

Even numbered minor versions (5.14, 5.16, 5.18) are production versions, and odd numbered minor versions (5.15, 5.17, 5.19) are development versions. Unless you want to try out an experimental feature, you probably never want to install a development version of Perl.

The Perl development team are called Perl 5 Porters, and their organization is described at *http://perldoc.perl.org/perlpolicy.html*. The organizational rules really just boil down to one: Larry is always right, even when he was wrong.

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?

Perl can be used for almost any coding problem, even ones which require integrating specialist C code for extra speed. As with any tool it can be used well or badly. Perl has many strengths, and a few weaknesses, precisely which areas are good and bad is often a personal choice.

When choosing a language you should also be influenced by the *resources*, *testing culture* and *community* which surrounds it.

For comparisons to a specific language it is often best to create a small project in both languages and compare the results, make sure to use all the *resources* of each language, as a language is far more than just it's syntax.

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?

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).

If you find that you need to speed up a specific part of a Perl application (not something you often need) you may want to use C, but you can access this from your Perl code with *perlxs*.

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

"Perl" is the name of the language. Only the "P" is capitalized. The name of the interpreter (the program which runs the Perl script) is "perl" with a lowercase "p".

You may or may not choose to follow this usage. But never write "PERL", because perl is not an acronym.

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*.

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://www.perl.org/about.html*

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

AUTHOR AND COPYRIGHT