NAME

perlfaq3 - Programming Tools

VERSION

version 5.021011

DESCRIPTION

This section of the FAQ answers questions related to programmer tools and programming support.

How do I do (anything)?

Have you looked at CPAN (see *perlfaq2*)? The chances are that someone has already written a module that can solve your problem. Have you read the appropriate manpages? Here's a brief index:

Basics

perldata - Perl data types

perlvar - Perl pre-defined variables

perlsyn - Perl syntax

perlop - Perl operators and precedence

perlsub - Perl subroutines

Execution

perlrun - how to execute the Perl interpreter

perldebug - Perl debugging

Functions

perlfunc - Perl builtin functions

Objects

perlref - Perl references and nested data structures

perlmod - Perl modules (packages and symbol tables)

perlobj - Perl objects

perltie - how to hide an object class in a simple variable

Data Structures

perlref - Perl references and nested data structures

perllol - Manipulating arrays of arrays in Perl

perldsc - Perl Data Structures Cookbook

Modules

perlmod - Perl modules (packages and symbol tables)

perlmodlib - constructing new Perl modules and finding existing ones

Regexes

perlre - Perl regular expressions

perlfunc - Perl builtin functions>

perlop - Perl operators and precedence

perllocale - Perl locale handling (internationalization and localization)

Moving to perl5

perltrap - Perl traps for the unwary

perl

Linking with C

perlxstut - Tutorial for writing XSUBs

perlxs - XS language reference manual

perlcall - Perl calling conventions from C

perlguts - Introduction to the Perl API

perlembed - how to embed perl in your C program

Various

http://www.cpan.org/misc/olddoc/FMTEYEWTK.tgz (not a man-page but still useful, a collection of various essays on Perl techniques)

A crude table of contents for the Perl manpage set is found in *perltoc*.

How can I use Perl interactively?

The typical approach uses the Perl debugger, described in the *perldebug(1)* manpage, on an "empty" program, like this:

```
perl -de 42
```

Now just type in any legal Perl code, and it will be immediately evaluated. You can also examine the symbol table, get stack backtraces, check variable values, set breakpoints, and other operations typically found in symbolic debuggers.

You can also use *Devel::REPL* which is an interactive shell for Perl, commonly known as a REPL - Read, Evaluate, Print, Loop. It provides various handy features.

How do I find which modules are installed on my system?

From the command line, you can use the `cpan` command's `-l` switch:

```
$ cpan -l
```

You can also use `cpan`'s `-a` switch to create an autobundle file that `CPAN.pm` understands and can use to re-install every module:

```
$ cpan -a
```

Inside a Perl program, you can use the *ExtUtils::Installed* module to show all installed distributions, although it can take awhile to do its magic. The standard library which comes with Perl just shows up as "Perl" (although you can get those with *Module::CoreList*).

```
use ExtUtils::Installed;

my $inst    = ExtUtils::Installed->new();
my @modules = $inst->modules();
```

If you want a list of all of the Perl module filenames, you can use *File::Find::Rule*:

```
use File::Find::Rule;

my @files = File::Find::Rule->
    extras({follow => 1})->
    file()->
    name( '*.pm' )->
    in( @INC )
    ;
```

If you do not have that module, you can do the same thing with *File::Find* which is part of the standard library:

```
use File::Find;
my @files;

find(
    {
    wanted => sub {
        push @files, $File::Find::fullname
        if -f $File::Find::fullname && /\.pm$/
    },
    follow => 1,
    follow_skip => 2,
    },
    @INC
);

print join "\n", @files;
```

If you simply need to check quickly to see if a module is available, you can check for its documentation. If you can read the documentation the module is most likely installed. If you cannot read the documentation, the module might not have any (in rare cases):

```
$ perldoc Module::Name
```

You can also try to include the module in a one-liner to see if perl finds it:

```
$ perl -MModule::Name -e1
```

(If you don't receive a "Can't locate ... in @INC" error message, then Perl found the module name you asked for.)

How do I debug my Perl programs?

(contributed by brian d foy)

Before you do anything else, you can help yourself by ensuring that you let Perl tell you about problem areas in your code. By turning on warnings and strictures, you can head off many problems before they get too big. You can find out more about these in *strict* and *warnings*.

```
#!/usr/bin/perl
use strict;
use warnings;
```

Beyond that, the simplest debugger is the `print` function. Use it to look at values as you run your program:

```
print STDERR "The value is [$value]\n";
```

The *Data::Dumper* module can pretty-print Perl data structures:

```
use Data::Dumper qw( Dumper );
print STDERR "The hash is " . Dumper( \%hash ) . "\n";
```

Perl comes with an interactive debugger, which you can start with the `-d` switch. It's fully explained in *perldebug*.

If you'd like a graphical user interface and you have *Tk*, you can use `ptkdb`. It's on CPAN and available for free.

If you need something much more sophisticated and controllable, Leon Brocard's *Devel::ebug* (which you can call with the `-D` switch as `-Debug`) gives you the programmatic hooks into everything you need to write your own (without too much pain and suffering).

You can also use a commercial debugger such as Affrus (Mac OS X), Komodo from Activestate (Windows and Mac OS X), or EPIC (most platforms).

How do I profile my Perl programs?

(contributed by brian d foy, updated Fri Jul 25 12:22:26 PDT 2008)

The `Devel` namespace has several modules which you can use to profile your Perl programs.

The *Devel::NYTProf* (New York Times Profiler) does both statement and subroutine profiling. It's available from CPAN and you also invoke it with the `-d` switch:

```
perl -d:NYTProf some_perl.pl
```

It creates a database of the profile information that you can turn into reports. The `nytprofhtml` command turns the data into an HTML report similar to the *Devel::Cover* report:

```
nytprofhtml
```

You might also be interested in using the *Benchmark* to measure and compare code snippets.

You can read more about profiling in *Programming Perl*, chapter 20, or *Mastering Perl*, chapter 5.

perldebguts documents creating a custom debugger if you need to create a special sort of profiler. brian d foy describes the process in *The Perl Journal*, "Creating a Perl Debugger", *http://www.ddj.com/184404522* , and "Profiling in Perl" *http://www.ddj.com/184404580* .

Perl.com has two interesting articles on profiling: "Profiling Perl", by Simon Cozens, *http://www.perl.com/lpt/a/850* and "Debugging and Profiling mod_perl Applications", by Frank Wiles, *http://www.perl.com/pub/a/2006/02/09/debug_mod_perl.html* .

Randal L. Schwartz writes about profiling in "Speeding up Your Perl Programs" for *Unix Review*, *http://www.stonehenge.com/merlyn/UnixReview/col49.html* , and "Profiling in Template Toolkit via Overriding" for *Linux Magazine*, *http://www.stonehenge.com/merlyn/LinuxMag/col75.html* .

How do I cross-reference my Perl programs?

The *B::Xref* module can be used to generate cross-reference reports for Perl programs.

```
perl -MO=Xref[,OPTIONS] scriptname.plx
```

Is there a pretty-printer (formatter) for Perl?

Perl::Tidy comes with a perl script *perltidy* which indents and reformats Perl scripts to make them easier to read by trying to follow the rules of the *perlstyle*. If you write Perl, or spend much time reading Perl, you will probably find it useful.

Of course, if you simply follow the guidelines in *perlstyle*, you shouldn't need to reformat. The habit of formatting your code as you write it will help prevent bugs. Your editor can and should help you with this. The perl-mode or newer cperl-mode for emacs can provide remarkable amounts of help with most (but not all) code, and even less programmable editors can provide significant assistance. Tom Christiansen and many other VI users swear by the following settings in vi and its clones:

```
set ai sw=4
map! ^O {^M}^[O^T
```

Put that in your *.exrc* file (replacing the caret characters with control characters) and away you go. In insert mode, ^T is for indenting, ^D is for undenting, and ^O is for blockdenting--as it were. A more complete example, with comments, can be found at *http://www.cpan.org/authors/id/TOMC/scripts/toms.exrc.gz*

Is there an IDE or Windows Perl Editor?

Perl programs are just plain text, so any editor will do.

If you're on Unix, you already have an IDE--Unix itself. The Unix philosophy is the philosophy of several small tools that each do one thing and do it well. It's like a carpenter's toolbox.

If you want an IDE, check the following (in alphabetical order, not order of preference):

Eclipse

http://e-p-i-c.sf.net/

The Eclipse Perl Integration Project integrates Perl editing/debugging with Eclipse.

Enginsite

http://www.enginsite.com/

Perl Editor by EngInSite is a complete integrated development environment (IDE) for creating, testing, and debugging Perl scripts; the tool runs on Windows 9x/NT/2000/XP or later.

IntelliJ IDEA

https://plugins.jetbrains.com/plugin/7796

Camelcade plugin provides Perl5 support in IntelliJ IDEA and other JetBrains IDEs.

Kephra

http://kephra.sf.net

GUI editor written in Perl using wxWidgets and Scintilla with lots of smaller features. Aims for a UI based on Perl principles like TIMTOWTDI and "easy things should be easy, hard things should be possible".

Komodo

http://www.ActiveState.com/Products/Komodo/

ActiveState's cross-platform (as of October 2004, that's Windows, Linux, and Solaris), multi-language IDE has Perl support, including a regular expression debugger and remote debugging.

Notepad++

http://notepad-plus.sourceforge.net/

Open Perl IDE

http://open-perl-ide.sourceforge.net/

Open Perl IDE is an integrated development environment for writing and debugging Perl scripts with ActiveState's ActivePerl distribution under Windows 95/98/NT/2000.

OptiPerl

http://www.optiperl.com/

OptiPerl is a Windows IDE with simulated CGI environment, including debugger and syntax-highlighting editor.

Padre

http://padre.perlide.org/

Padre is cross-platform IDE for Perl written in Perl using wxWidgets to provide a native look

and feel. It's open source under the Artistic License. It is one of the newer Perl IDEs.

PerlBuilder

http://www.solutionsoft.com/perl.htm

PerlBuilder is an integrated development environment for Windows that supports Perl development.

visiPerl+

http://helpconsulting.net/visiperl/index.html

From Help Consulting, for Windows.

Visual Perl

http://www.activestate.com/Products/Visual_Perl/

Visual Perl is a Visual Studio.NET plug-in from ActiveState.

Zeus

http://www.zeusedit.com/lookmain.html

Zeus for Windows is another Win32 multi-language editor/IDE that comes with support for Perl.

For editors: if you're on Unix you probably have vi or a vi clone already, and possibly an emacs too, so you may not need to download anything. In any emacs the cperl-mode (M-x cperl-mode) gives you perhaps the best available Perl editing mode in any editor.

If you are using Windows, you can use any editor that lets you work with plain text, such as NotePad or WordPad. Word processors, such as Microsoft Word or WordPerfect, typically do not work since they insert all sorts of behind-the-scenes information, although some allow you to save files as "Text Only". You can also download text editors designed specifically for programming, such as Textpad (*http://www.textpad.com/*) and UltraEdit (*http://www.ultraedit.com/*), among others.

If you are using MacOS, the same concerns apply. MacPerl (for Classic environments) comes with a simple editor. Popular external editors are BBEdit (*http://www.barebones.com/products/bbedit/*) or Alpha (*http://www.his.com/~jguyer/Alpha/Alpha8.html*). MacOS X users can use Unix editors as well.

GNU Emacs

http://www.gnu.org/software/emacs/windows/ntemacs.html

MicroEMACS

http://www.microemacs.de/

XEmacs

http://www.xemacs.org/Download/index.html

Jed

http://space.mit.edu/~davis/jed/

or a vi clone such as

Vim

http://www.vim.org/

Vile

http://dickey.his.com/vile/vile.html

The following are Win32 multilanguage editor/IDEs that support Perl:

MultiEdit

http://www.MultiEdit.com/

SlickEdit

http://www.slickedit.com/

ConTEXT

http://www.contexteditor.org/

There is also a toyedit Text widget based editor written in Perl that is distributed with the Tk module on CPAN. The ptkdb (*http://ptkdb.sourceforge.net/*) is a Perl/Tk-based debugger that acts as a development environment of sorts. Perl Composer (*http://perlcomposer.sourceforge.net/*) is an IDE for Perl/Tk GUI creation.

In addition to an editor/IDE you might be interested in a more powerful shell environment for Win32. Your options include

bash

from the Cygwin package (*http://cygwin.com/*)

zsh

http://www.zsh.org/

Cygwin is covered by the GNU General Public License (but that shouldn't matter for Perl use). Cygwin contains (in addition to the shell) a comprehensive set of standard Unix toolkit utilities.

BBEdit and TextWrangler

are text editors for OS X that have a Perl sensitivity mode (*http://www.barebones.com/*).

Where can I get Perl macros for vi?

For a complete version of Tom Christiansen's vi configuration file, see *http://www.cpan.org/authors/Tom_Christiansen/scripts/toms.exrc.gz* , the standard benchmark file for vi emulators. The file runs best with nvi, the current version of vi out of Berkeley, which incidentally can be built with an embedded Perl interpreter--see *http://www.cpan.org/src/misc/* .

Where can I get perl-mode or cperl-mode for emacs?

Since Emacs version 19 patchlevel 22 or so, there have been both a perl-mode.el and support for the Perl debugger built in. These should come with the standard Emacs 19 distribution.

Note that the perl-mode of emacs will have fits with `"main'foo"` (single quote), and mess up the indentation and highlighting. You are probably using `"main::foo"` in new Perl code anyway, so this shouldn't be an issue.

For CPerlMode, see *http://www.emacswiki.org/cgi-bin/wiki/CPerlMode*

How can I use curses with Perl?

The Curses module from CPAN provides a dynamically loadable object module interface to a curses library. A small demo can be found at the directory *http://www.cpan.org/authors/Tom_Christiansen/scripts/rep.gz* ; this program repeats a command and updates the screen as needed, rendering **rep ps axu** similar to **top**.

How can I write a GUI (X, Tk, Gtk, etc.) in Perl?

(contributed by Ben Morrow)

There are a number of modules which let you write GUIs in Perl. Most GUI toolkits have a perl interface: an incomplete list follows.

Tk

This works under Unix and Windows, and the current version doesn't look half as bad under

Windows as it used to. Some of the gui elements still don't 'feel' quite right, though. The interface is very natural and 'perlish', making it easy to use in small scripts that just need a simple gui. It hasn't been updated in a while.

Wx

This is a Perl binding for the cross-platform wxWidgets toolkit (*http://www.wxwidgets.org*). It works under Unix, Win32 and Mac OS X, using native widgets (Gtk under Unix). The interface follows the C++ interface closely, but the documentation is a little sparse for someone who doesn't know the library, mostly just referring you to the C++ documentation.

Gtk and Gtk2

These are Perl bindings for the Gtk toolkit (*http://www.gtk.org*). The interface changed significantly between versions 1 and 2 so they have separate Perl modules. It runs under Unix, Win32 and Mac OS X (currently it requires an X server on Mac OS, but a 'native' port is underway), and the widgets look the same on every platform: i.e., they don't match the native widgets. As with Wx, the Perl bindings follow the C API closely, and the documentation requires you to read the C documentation to understand it.

Win32::GUI

This provides access to most of the Win32 GUI widgets from Perl. Obviously, it only runs under Win32, and uses native widgets. The Perl interface doesn't really follow the C interface: it's been made more Perlish, and the documentation is pretty good. More advanced stuff may require familiarity with the C Win32 APIs, or reference to MSDN.

CamelBones

CamelBones (*http://camelbones.sourceforge.net*) is a Perl interface to Mac OS X's Cocoa GUI toolkit, and as such can be used to produce native GUIs on Mac OS X. It's not on CPAN, as it requires frameworks that CPAN.pm doesn't know how to install, but installation is via the standard OSX package installer. The Perl API is, again, very close to the ObjC API it's wrapping, and the documentation just tells you how to translate from one to the other.

Qt

There is a Perl interface to TrollTech's Qt toolkit, but it does not appear to be maintained.

Athena

Sx is an interface to the Athena widget set which comes with X, but again it appears not to be much used nowadays.

How can I make my Perl program run faster?

The best way to do this is to come up with a better algorithm. This can often make a dramatic difference. Jon Bentley's book *Programming Pearls* (that's not a misspelling!) has some good tips on optimization, too. Advice on benchmarking boils down to: benchmark and profile to make sure you're optimizing the right part, look for better algorithms instead of microtuning your code, and when all else fails consider just buying faster hardware. You will probably want to read the answer to the earlier question "How do I profile my Perl programs?" if you haven't done so already.

A different approach is to autoload seldom-used Perl code. See the AutoSplit and AutoLoader modules in the standard distribution for that. Or you could locate the bottleneck and think about writing just that part in C, the way we used to take bottlenecks in C code and write them in assembler. Similar to rewriting in C, modules that have critical sections can be written in C (for instance, the PDL module from CPAN).

If you're currently linking your perl executable to a shared *libc.so*, you can often gain a 10-25% performance benefit by rebuilding it to link with a static libc.a instead. This will make a bigger perl executable, but your Perl programs (and programmers) may thank you for it. See the *INSTALL* file in the source distribution for more information.

The undump program was an ancient attempt to speed up Perl program by storing the already-compiled form to disk. This is no longer a viable option, as it only worked on a few architectures, and wasn't a good solution anyway.

How can I make my Perl program take less memory?

When it comes to time-space tradeoffs, Perl nearly always prefers to throw memory at a problem. Scalars in Perl use more memory than strings in C, arrays take more than that, and hashes use even more. While there's still a lot to be done, recent releases have been addressing these issues. For example, as of 5.004, duplicate hash keys are shared amongst all hashes using them, so require no reallocation.

In some cases, using substr() or vec() to simulate arrays can be highly beneficial. For example, an array of a thousand booleans will take at least 20,000 bytes of space, but it can be turned into one 125-byte bit vector--a considerable memory savings. The standard Tie::SubstrHash module can also help for certain types of data structure. If you're working with specialist data structures (matrices, for instance) modules that implement these in C may use less memory than equivalent Perl modules.

Another thing to try is learning whether your Perl was compiled with the system malloc or with Perl's builtin malloc. Whichever one it is, try using the other one and see whether this makes a difference. Information about malloc is in the *INSTALL* file in the source distribution. You can find out whether you are using perl's malloc by typing `perl -V:usemymalloc`.

Of course, the best way to save memory is to not do anything to waste it in the first place. Good programming practices can go a long way toward this:

Don't slurp!

Don't read an entire file into memory if you can process it line by line. Or more concretely, use a loop like this:

```
#
# Good Idea
#
while (my $line = <$file_handle>) {
   # ...
}
```

instead of this:

```
#
# Bad Idea
#
my @data = <$file_handle>;
foreach (@data) {
    # ...
}
```

When the files you're processing are small, it doesn't much matter which way you do it, but it makes a huge difference when they start getting larger.

Use map and grep selectively

Remember that both map and grep expect a LIST argument, so doing this:

```
@wanted = grep {/pattern/} <$file_handle>;
```

will cause the entire file to be slurped. For large files, it's better to loop:

```
while (<$file_handle>) {
        push(@wanted, $_) if /pattern/;
}
```

Avoid unnecessary quotes and stringification

Don't quote large strings unless absolutely necessary:

```
my $copy = "$large_string";
```

makes 2 copies of $large_string (one for $copy and another for the quotes), whereas

```
my $copy = $large_string;
```

only makes one copy.

Ditto for stringifying large arrays:

```
{
local $, = "\n";
print @big_array;
}
```

is much more memory-efficient than either

```
print join "\n", @big_array;
```

or

```
{
local $" = "\n";
print "@big_array";
}
```

Pass by reference

Pass arrays and hashes by reference, not by value. For one thing, it's the only way to pass multiple lists or hashes (or both) in a single call/return. It also avoids creating a copy of all the contents. This requires some judgement, however, because any changes will be propagated back to the original data. If you really want to mangle (er, modify) a copy, you'll have to sacrifice the memory needed to make one.

Tie large variables to disk

For "big" data stores (i.e. ones that exceed available memory) consider using one of the DB modules to store it on disk instead of in RAM. This will incur a penalty in access time, but that's probably better than causing your hard disk to thrash due to massive swapping.

Is it safe to return a reference to local or lexical data?

Yes. Perl's garbage collection system takes care of this so everything works out right.

```
sub makeone {
    my @a = ( 1 .. 10 );
    return \@a;
}

for ( 1 .. 10 ) {
    push @many, makeone();
}

print $many[4][5], "\n";

print "@many\n";
```

How can I free an array or hash so my program shrinks?

(contributed by Michael Carman)

You usually can't. Memory allocated to lexicals (i.e. my() variables) cannot be reclaimed or reused even if they go out of scope. It is reserved in case the variables come back into scope. Memory allocated to global variables can be reused (within your program) by using undef() and/or delete().

On most operating systems, memory allocated to a program can never be returned to the system. That's why long-running programs sometimes re- exec themselves. Some operating systems (notably, systems that use mmap(2) for allocating large chunks of memory) can reclaim memory that is no longer used, but on such systems, perl must be configured and compiled to use the OS's malloc, not perl's.

In general, memory allocation and de-allocation isn't something you can or should be worrying about much in Perl.

See also "How can I make my Perl program take less memory?"

How can I make my CGI script more efficient?

Beyond the normal measures described to make general Perl programs faster or smaller, a CGI program has additional issues. It may be run several times per second. Given that each time it runs it will need to be re-compiled and will often allocate a megabyte or more of system memory, this can be a killer. Compiling into C **isn't going to help you** because the process start-up overhead is where the bottleneck is.

There are three popular ways to avoid this overhead. One solution involves running the Apache HTTP server (available from *http://www.apache.org/*) with either of the mod_perl or mod_fastcgi plugin modules.

With mod_perl and the Apache::Registry module (distributed with mod_perl), httpd will run with an embedded Perl interpreter which pre-compiles your script and then executes it within the same address space without forking. The Apache extension also gives Perl access to the internal server API, so modules written in Perl can do just about anything a module written in C can. For more on mod_perl, see *http://perl.apache.org/*

With the FCGI module (from CPAN) and the mod_fastcgi module (available from *http://www.fastcgi.com/*) each of your Perl programs becomes a permanent CGI daemon process.

Finally, *Plack* is a Perl module and toolkit that contains PSGI middleware, helpers and adapters to web servers, allowing you to easily deploy scripts which can continue running, and provides flexibility with regards to which web server you use. It can allow existing CGI scripts to enjoy this flexibility and performance with minimal changes, or can be used along with modern Perl web frameworks to make writing and deploying web services with Perl a breeze.

These solutions can have far-reaching effects on your system and on the way you write your CGI programs, so investigate them with care.

See also *http://www.cpan.org/modules/by-category/15_World_Wide_Web_HTML_HTTP_CGI/* .

How can I hide the source for my Perl program?

Delete it. :-) Seriously, there are a number of (mostly unsatisfactory) solutions with varying levels of "security".

First of all, however, you *can't* take away read permission, because the source code has to be readable in order to be compiled and interpreted. (That doesn't mean that a CGI script's source is readable by people on the web, though--only by people with access to the filesystem.) So you have to leave the permissions at the socially friendly 0755 level.

Some people regard this as a security problem. If your program does insecure things and relies on people not knowing how to exploit those insecurities, it is not secure. It is often possible for someone

to determine the insecure things and exploit them without viewing the source. Security through obscurity, the name for hiding your bugs instead of fixing them, is little security indeed.

You can try using encryption via source filters (Starting from Perl 5.8 the Filter::Simple and Filter::Util::Call modules are included in the standard distribution), but any decent programmer will be able to decrypt it. You can try using the byte code compiler and interpreter described later in *perlfaq3*, but the curious might still be able to de-compile it. You can try using the native-code compiler described later, but crackers might be able to disassemble it. These pose varying degrees of difficulty to people wanting to get at your code, but none can definitively conceal it (true of every language, not just Perl).

It is very easy to recover the source of Perl programs. You simply feed the program to the perl interpreter and use the modules in the B:: hierarchy. The B::Deparse module should be able to defeat most attempts to hide source. Again, this is not unique to Perl.

If you're concerned about people profiting from your code, then the bottom line is that nothing but a restrictive license will give you legal security. License your software and pepper it with threatening statements like "This is unpublished proprietary software of XYZ Corp. Your access to it does not give you permission to use it blah blah blah." We are not lawyers, of course, so you should see a lawyer if you want to be sure your license's wording will stand up in court.

How can I compile my Perl program into byte code or C?

(contributed by brian d foy)

In general, you can't do this. There are some things that may work for your situation though. People usually ask this question because they want to distribute their works without giving away the source code, and most solutions trade disk space for convenience. You probably won't see much of a speed increase either, since most solutions simply bundle a Perl interpreter in the final product (but see *How can I make my Perl program run faster?*).

The Perl Archive Toolkit (*http://par.perl.org/*) is Perl's analog to Java's JAR. It's freely available and on CPAN (*http://search.cpan.org/dist/PAR/*).

There are also some commercial products that may work for you, although you have to buy a license for them.

The Perl Dev Kit (*http://www.activestate.com/Products/Perl_Dev_Kit/*) from ActiveState can "Turn your Perl programs into ready-to-run executables for HP-UX, Linux, Solaris and Windows."

Perl2Exe (*http://www.indigostar.com/perl2exe.htm*) is a command line program for converting perl scripts to executable files. It targets both Windows and Unix platforms.

How can I get #!perl to work on [MS-DOS,NT,...]?

For OS/2 just use

```
extproc perl -S -your_switches
```

as the first line in `*.cmd` file (`-S` due to a bug in cmd.exe's "extproc" handling). For DOS one should first invent a corresponding batch file and codify it in `ALTERNATE_SHEBANG` (see the *dosish.h* file in the source distribution for more information).

The Win95/NT installation, when using the ActiveState port of Perl, will modify the Registry to associate the `.pl` extension with the perl interpreter. If you install another port, perhaps even building your own Win95/NT Perl from the standard sources by using a Windows port of gcc (e.g., with cygwin or mingw32), then you'll have to modify the Registry yourself. In addition to associating `.pl` with the interpreter, NT people can use: `SET PATHEXT=%PATHEXT%;.PL` to let them run the program `install-linux.pl` merely by typing `install-linux`.

Under "Classic" MacOS, a perl program will have the appropriate Creator and Type, so that

double-clicking them will invoke the MacPerl application. Under Mac OS X, clickable apps can be made from any `#!` script using Wil Sanchez' DropScript utility: *http://www.wsanchez.net/software/* .

IMPORTANT!: Whatever you do, PLEASE don't get frustrated, and just throw the perl interpreter into your cgi-bin directory, in order to get your programs working for a web server. This is an EXTREMELY big security risk. Take the time to figure out how to do it correctly.

Can I write useful Perl programs on the command line?

Yes. Read *perlrun* for more information. Some examples follow. (These assume standard Unix shell quoting rules.)

```
# sum first and last fields
perl -lane 'print $F[0] + $F[-1]' *

# identify text files
perl -le 'for(@ARGV) {print if -f && -T _}' *

# remove (most) comments from C program
perl -0777 -pe 's{/\*.*?\*/}{}gs' foo.c

# make file a month younger than today, defeating reaper daemons
perl -e '$X=24*60*60; utime(time(),time() + 30 * $X,@ARGV)' *

# find first unused uid
perl -le '$i++ while getpwuid($i); print $i'

# display reasonable manpath
echo $PATH | perl -nl -072 -e '
s![^/+]*$!man!&&-d&&!$s{$_}++&&push@m,$_;END{print"@m"}'
```

OK, the last one was actually an Obfuscated Perl Contest entry. :-)

Why don't Perl one-liners work on my DOS/Mac/VMS system?

The problem is usually that the command interpreters on those systems have rather different ideas about quoting than the Unix shells under which the one-liners were created. On some systems, you may have to change single-quotes to double ones, which you must *NOT* do on Unix or Plan9 systems. You might also have to change a single % to a %%.

For example:

```
# Unix (including Mac OS X)
perl -e 'print "Hello world\n"'

# DOS, etc.
perl -e "print \"Hello world\n\""

# Mac Classic
print "Hello world\n"
 (then Run "Myscript" or Shift-Command-R)

# MPW
perl -e 'print "Hello world\n"'

# VMS
```

```
perl -e "print ""Hello world\n"""
```

The problem is that none of these examples are reliable: they depend on the command interpreter. Under Unix, the first two often work. Under DOS, it's entirely possible that neither works. If 4DOS was the command shell, you'd probably have better luck like this:

```
perl -e "print <Ctrl-x>"Hello world\n<Ctrl-x>""
```

Under the Mac, it depends which environment you are using. The MacPerl shell, or MPW, is much like Unix shells in its support for several quoting variants, except that it makes free use of the Mac's non-ASCII characters as control characters.

Using qq(), q(), and qx(), instead of "double quotes", 'single quotes', and `backticks`, may make one-liners easier to write.

There is no general solution to all of this. It is a mess.

[Some of this answer was contributed by Kenneth Albanowski.]

Where can I learn about CGI or Web programming in Perl?

For modules, get the CGI or LWP modules from CPAN. For textbooks, see the two especially dedicated to web stuff in the question on books. For problems and questions related to the web, like "Why do I get 500 Errors" or "Why doesn't it run from the browser right when it runs fine on the command line", see the troubleshooting guides and references in *perlfaq9* or in the CGI MetaFAQ:

```
L<http://www.perl.org/CGI_MetaFAQ.html>
```

Looking in to *Plack* and modern Perl web frameworks is highly recommended, though; web programming in Perl has evolved a long way from the old days of simple CGI scripts.

Where can I learn about object-oriented Perl programming?

A good place to start is *perlootut*, and you can use *perlobj* for reference.

A good book on OO on Perl is the "Object-Oriented Perl" by Damian Conway from Manning Publications, or "Intermediate Perl" by Randal Schwartz, brian d foy, and Tom Phoenix from O'Reilly Media.

Where can I learn about linking C with Perl?

If you want to call C from Perl, start with *perlxstut*, moving on to *perlxs*, *xsubpp*, and *perlguts*. If you want to call Perl from C, then read *perlembed*, *perlcall*, and *perlguts*. Don't forget that you can learn a lot from looking at how the authors of existing extension modules wrote their code and solved their problems.

You might not need all the power of XS. The Inline::C module lets you put C code directly in your Perl source. It handles all the magic to make it work. You still have to learn at least some of the perl API but you won't have to deal with the complexity of the XS support files.

I've read perlembed, perlguts, etc., but I can't embed perl in my C program; what am I doing wrong?

Download the ExtUtils::Embed kit from CPAN and run `make test'. If the tests pass, read the pods again and again and again. If they fail, see *perlbug* and send a bug report with the output of `make test TEST_VERBOSE=1` along with `perl -V`.

When I tried to run my script, I got this message. What does it mean?

A complete list of Perl's error messages and warnings with explanatory text can be found in *perldiag*. You can also use the splain program (distributed with Perl) to explain the error messages:

```
perl program 2>diag.out
```

```
splain [-v] [-p] diag.out
```

or change your program to explain the messages for you:

```
use diagnostics;
```

or

```
use diagnostics -verbose;
```

What's MakeMaker?

(contributed by brian d foy)

The *ExtUtils::MakeMaker* module, better known simply as "MakeMaker", turns a Perl script, typically called `Makefile.PL`, into a Makefile. The Unix tool `make` uses this file to manage dependencies and actions to process and install a Perl distribution.

AUTHOR AND COPYRIGHT