

NAME

perlfreq3 - Programming Tools (\$Revision: 1.47 \$, \$Date: 2005/03/27 07:21:22 \$)

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 *perlfreq2*)? 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, perlvar, perlsyn, perlop, perlsub
Execution       perlrun, perldebug
Functions       perlfunc
Objects         perlref, perlmod, perlobj, perltie
Data Structures perlref, perllo1, perldsc
Modules         perlmod, perlmodlib, perlsub
Regexes        perlre, perlfunc, perlop, perllocale
Moving to perl5 perltrap, perl
Linking w/C     perlxs, perlcall, perlguts, perlembed
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.

Is there a Perl shell?

The psh (Perl sh) is currently at version 1.8. The Perl Shell is a shell that combines the interactive nature of a Unix shell with the power of Perl. The goal is a full featured shell that behaves as expected for normal shell activity and uses Perl syntax and functionality for control-flow statements and other things. You can get psh at <http://www.focusresearch.com/gregor/psh/>.

Zoidberg is a similar project and provides a shell written in perl, configured in perl and operated in perl. It is intended as a login shell and development environment. It can be found at <http://zoidberg.sf.net/> or your local CPAN mirror.

The Shell.pm module (distributed with Perl) makes Perl try commands which aren't part of the Perl language as shell commands. *perlsh* from the source distribution is simplistic and uninteresting, but may still be what you want.

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

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->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 sub { push @files, $File::Find::name if -f _ && /\.pm$/ },
      @INC;

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

If you simply need to quickly check 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).

```
prompt% 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
```

How do I debug my Perl programs?

Have you tried `use warnings` or used `-w`? They enable warnings to detect dubious practices.

Have you tried `use strict`? It prevents you from using symbolic references, makes you predeclare any subroutines that you call as bare words, and (probably most importantly) forces you to predeclare your variables with `my`, `our`, or `use vars`.

Did you check the return values of each and every system call? The operating system (and thus Perl) tells you whether they worked, and if not why.

```
open(FH, "> /etc/cantwrite")
  or die "Couldn't write to /etc/cantwrite: $!\n";
```

Did you read *perltrap*? It's full of gotchas for old and new Perl programmers and even has sections for those of you who are upgrading from languages like *awk* and *C*.

Have you tried the Perl debugger, described in *perldebug*? You can step through your program and see what it's doing and thus work out why what it's doing isn't what it should be doing.

How do I profile my Perl programs?

You should get the `Devel::DProf` module from the standard distribution (or separately on CPAN) and also use `Benchmark.pm` from the standard distribution. The `Benchmark` module lets you time specific portions of your code, while `Devel::DProf` gives detailed breakdowns of where your code spends its time.

Here's a sample use of `Benchmark`:

```
use Benchmark;

@junk = `cat /etc/motd`;
$count = 10_000;

timethese($count, {
    'map' => sub { my @a = @junk;
        map { s/a/b/ } @a;
        return @a },
    'for' => sub { my @a = @junk;
        for (@a) { s/a/b/ };
        return @a },
});
```

This is what it prints (on one machine--your results will be dependent on your hardware, operating system, and the load on your machine):

```
Benchmark: timing 10000 iterations of for, map...
   for:  4 secs ( 3.97 usr  0.01 sys =  3.98 cpu)
   map:  6 secs ( 4.97 usr  0.00 sys =  4.97 cpu)
```

Be aware that a good benchmark is very hard to write. It only tests the data you give it and proves little about the differing complexities of contrasting algorithms.

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?

Perltidy is a Perl script 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 scripts, or spend much time reading them, you will probably find it useful. It is available at <http://perltidy.sourceforge.net>

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 unindenting, 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>

The a2ps <http://www-inf.enst.fr/%7Edemaille/a2ps/black+white.ps.gz> does lots of things related to generating nicely printed output of documents, as does `enscript` at <http://people.ssh.fi/mtr/genscript/>.

Is there a ctags for Perl?

Recent versions of ctags do much more than older versions did. EXUBERANT CTAGS is available from <http://ctags.sourceforge.net/> and does a good job of making tags files for perl code.

There is also a simple one at <http://www.cpan.org/authors/id/TOMC/scripts/ptags.gz> which may do the trick. It can be easy to hack this into what you want.

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

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

The website for the project is <http://e-p-i-c.sf.net/>

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 (<http://www.ActiveState.com/Products/Komodo/>).

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/>) is a Windows IDE with simulated CGI environment, including debugger and syntax highlighting editor.

PerlBuilder

(<http://www.solutionsoft.com/perl.htm>) is an integrated development environment for Windows that supports Perl development.

visiPerl+

(<http://helpconsulting.net/visiperl/>) 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.

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.bbedit.com/>) 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

Elvis

<ftp://ftp.cs.pdx.edu/pub/elvis/> <http://www.fh-wedel.de/elvis/>

Vile

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

Vim

<http://www.vim.org/>

For vi lovers in general, Windows or elsewhere:

<http://www.thomer.com/thomer/vi/vi.html>

nvi (<http://www.bostic.com/vi/> , available from CPAN in src/misc/) is yet another vi clone, unfortunately not available for Windows, but in UNIX platforms you might be interested in trying it out, firstly because strictly speaking it is not a vi clone, it is the real vi, or the new incarnation of it, and secondly because you can embed Perl inside it to use Perl as the scripting language. nvi is not alone in this, though: at least also vim and vile offer an embedded Perl.

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

Codewright

<http://www.borland.com/codewright/>

MultiEdit

<http://www.MultiEdit.com/>

SlickEdit

<http://www.slickedit.com/>

There is also a toyedit Text widget based editor written in Perl that is distributed with the Tk module on CPAN. The ptkdb (<http://world.std.com/~aep/ptkdb/>) 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://sources.redhat.com/cygwin/>)

Ksh

from the MKS Toolkit (<http://www.mks.com/>), or the Bourne shell of the U/WIN environment (<http://www.research.att.com/sw/tools/uwin/>)

Tcsh

<ftp://ftp.astron.com/pub/tcsh/> , see also <http://www.primite.wisc.edu/software/csh-tcsh-book/>

Zsh

<ftp://ftp.blarg.net/users/amol/zsh/> , see also <http://www.zsh.org/>

MKS and U/WIN are commercial (U/WIN is free for educational and research purposes), Cygwin is covered by the GNU Public License (but that shouldn't matter for Perl use). The Cygwin, MKS, and U/WIN all contain (in addition to the shells) a comprehensive set of standard UNIX toolkit utilities.

If you're transferring text files between Unix and Windows using FTP be sure to transfer them in ASCII mode so the ends of lines are appropriately converted.

On Mac OS the MacPerl Application comes with a simple 32k text editor that behaves like a rudimentary IDE. In contrast to the MacPerl Application the MPW Perl tool can make use of the MPW Shell itself as an editor (with no 32k limit).

Affrus

is a full Perl development environment with full debugger support (<http://www.latenightsw.com>).

Alpha

is an editor, written and extensible in Tcl, that nonetheless has built in support for several popular markup and programming languages including Perl and HTML (<http://www.his.com/~jguyer/Alpha/Alpha8.html>).

BBEdit and BBEdit Lite

are text editors for Mac OS that have a Perl sensitivity mode (<http://web.barebones.com/>).

Pepper and Pe are programming language sensitive text editors for Mac OS X and BeOS respectively (<http://www.hekkelman.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 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.

In the Perl source directory, you'll find a directory called "emacs", which contains a cperl-mode that color-codes keywords, provides context-sensitive help, and other nifty things.

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.

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 use X or Tk with Perl?

Tk is a completely Perl-based, object-oriented interface to the Tk toolkit that doesn't force you to use Tcl just to get at Tk. Sx is an interface to the Athena Widget set. Both are available from CPAN. See the directory http://www.cpan.org/modules/by-category/08_User_Interfaces/

Invaluable for Perl/Tk programming are the Perl/Tk FAQ at

<http://phaseit.net/claird/comp.lang.perl.tk/ptkFAQ.html> , the Perl/Tk Reference Guide available at http://www.cpan.org/authors/Stephen_O_Lidie/ , and the online manpages at <http://www-users.cs.umn.edu/%7Eamundson/perl/perlTk/toc.html> .

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 (<FILE>) {
    # ...
}
```

instead of this:

```
#
# Bad Idea
#
@data = <FILE>;
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>;
```

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

```
while (<FILE>) {
    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 judgment, 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()ing 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 two 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.

Both of 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 http://www.cpan.org/modules/by-category/15_World_Wide_Web_HTML_HTTP_CGI/ .

A non-free, commercial product, ``The Velocity Engine for Perl'', (<http://www.binevolve.com/> or <http://www.binevolve.com/velocigen/>) might also be worth looking at. It will allow you to increase the performance of your Perl programs, running programs up to 25 times faster than normal CGI Perl when running in persistent Perl mode or 4 to 5 times faster without any modification to your existing CGI programs. Fully functional evaluation copies are available from the web site.

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 below, but the curious might still be able to de-compile it. You can try using the native-code compiler described below, 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/index.cgi>) is Perl's analog to Java's JAR. It's freely available and on CPAN (<http://search.cpan.org/dist/PAR/>).

The `B::*` namespace, often called "the Perl compiler", but is really a way for Perl programs to peek at its innards rather than create pre-compiled versions of your program. However, the `B::Bytecode` module can turn your script into a bytecode format that could be loaded later by the `ByteLoader` module and executed as a regular Perl script.

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 compile Perl into Java?

You can also integrate Java and Perl with the Perl Resource Kit from O'Reilly Media. See <http://www.oreilly.com/catalog/prkunix/>.

Perl 5.6 comes with Java Perl Lingo, or JPL. JPL, still in development, allows Perl code to be called from Java. See `jpl/README` in the Perl source tree.

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

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

Mac
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:

http://www.perl.org/CGI_MetaFAQ.html

Where can I learn about object-oriented Perl programming?

A good place to start is *perltoot*, and you can use *perlobj*, *perlboot*, *perltoot*, *perltoc*, and *perlbot* for reference. (If you are using really old Perl, you may not have all of these, try <http://www.perldoc.com/>, but consider upgrading your perl.)

A good book on OO on Perl is the "Object-Oriented Perl" by Damian Conway from Manning Publications, <http://www.manning.com/Conway/index.html>

Where can I learn about linking C with Perl? [h2xs, xsubpp]

If you want to call C from Perl, start with *perlxsut*, moving on to *perlxs*, *xsubpp*, and *perlguts*. If you want to call Perl from C, then read *perlembed*, *perllcall*, 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.

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?

This module (part of the standard Perl distribution) is designed to write a Makefile for an extension module from a Makefile.PL. For more information, see *ExtUtils::MakeMaker*.

AUTHOR AND COPYRIGHT

Copyright (c) 1997-2005 Tom Christiansen, Nathan Torkington, and other authors as noted. All rights reserved.

This documentation is free; you can redistribute it and/or modify it under the same terms as Perl itself.

Irrespective of its distribution, all code examples here are in the public domain. You are permitted and encouraged to use this code and any derivatives thereof in your own programs for fun or for profit as you see fit. A simple comment in the code giving credit to the FAQ would be courteous but is not required.