

## NAME

perl – Practical Extraction and Report Language

## SYNOPSIS

```
perl [ -sTuU ] [ -hv ] [ -V[:configvar] ]
      [ -cw ] [ -d[:debugger] ] [ -D[number/list] ]
      [ -pna ] [ -Fpattern ] [ -l[octal] ] [ -O[octal] ]
      [ -Idir ] [ -m[-]module ] [ -M[-]'module...' ] [ -P ] [ -S ]
      [ -x[dir] ] [ -i[extension] ]
      [ -e 'command' ] [ -- ] [ programfile ] [ argument ]...
```

For ease of access, the Perl manual has been split up into a number of sections:

perl	Perl overview (this section)
perldelta	Perl changes since previous version
perlfaq	Perl frequently asked questions
perltoc	Perl documentation table of contents
perldata	Perl data structures
perlsyn	Perl syntax
perlop	Perl operators and precedence
perlre	Perl regular expressions
perlrun	Perl execution and options
perlfunc	Perl builtin functions
perlvar	Perl predefined variables
perlsub	Perl subroutines
perlmod	Perl modules: how they work
perlmodlib	Perl modules: how to write and use
perlmodinstall	Perl modules: how to install from CPAN
perlform	Perl formats
perllocale	Perl locale support
perlref	Perl references
perldsc	Perl data structures intro
perllo1	Perl data structures: lists of lists
perltoot	Perl OO tutorial
perlobj	Perl objects
perltie	Perl objects hidden behind simple variables
perlbot	Perl OO tricks and examples
perlipc	Perl interprocess communication
perldebug	Perl debugging
perldiag	Perl diagnostic messages
perlsec	Perl security
perltrap	Perl traps for the unwary
perlport	Perl portability guide
perlstyle	Perl style guide
perlpod	Perl plain old documentation
perlbook	Perl book information

<code>perlembed</code>	Perl ways to embed perl in your C or C++ applica
<code>perlpio</code>	Perl internal IO abstraction interface
<code>perlxs</code>	Perl XS application programming interface
<code>perlxsut</code>	Perl XS tutorial
<code>perlguts</code>	Perl internal functions for those doing extensio
<code>perlcall</code>	Perl calling conventions from C
<code>perlhst</code>	Perl history records

(If you're intending to read these straight through for the first time, the suggested order will tend to reduce the number of forward references.)

By default, all of the above manpages are installed in the `/usr/local/man/` directory.

Extensive additional documentation for Perl modules is available. The default configuration for perl will place this additional documentation in the `/usr/local/lib/perl5/man` directory (or else in the `man` subdirectory of the Perl library directory). Some of this additional documentation is distributed standard with Perl, but you'll also find documentation for third-party modules there.

You should be able to view Perl's documentation with your `man(1)` program by including the proper directories in the appropriate start-up files, or in the `MANPATH` environment variable. To find out where the configuration has installed the manpages, type:

```
perl -V:man.dir
```

If the directories have a common stem, such as `/usr/local/man/man1` and `/usr/local/man/man3`, you need only to add that stem (`/usr/local/man`) to your `man(1)` configuration files or your `MANPATH` environment variable. If they do not share a stem, you'll have to add both stems.

If that doesn't work for some reason, you can still use the supplied `perldoc` script to view module information. You might also look into getting a replacement man program.

If something strange has gone wrong with your program and you're not sure where you should look for help, try the `-w` switch first. It will often point out exactly where the trouble is.

## DESCRIPTION

Perl is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

Perl combines (in the author's opinion, anyway) some of the best features of C, **sed**, **awk**, and **sh**, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of **csh**, Pascal, and even BASIC-PLUS.) Expression syntax corresponds quite closely to C expression syntax. Unlike most Unix utilities, Perl does not arbitrarily limit the size of your data—if you've got the memory, Perl can slurp in your whole file as a single string. Recursion is of unlimited depth. And the tables used by hashes (previously called "associative arrays") grow as necessary to prevent degraded performance. Perl uses sophisticated pattern matching techniques to scan large amounts of data very quickly. Although

optimized for scanning text, Perl can also deal with binary data, and can make dbm files look like hashes. Setuid Perl scripts are safer than C programs through a dataflow tracing mechanism which prevents many stupid security holes.

If you have a problem that would ordinarily use **sed** or **awk** or **sh**, but it exceeds their capabilities or must run a little faster, and you don't want to write the silly thing in C, then Perl may be for you. There are also translators to turn your **sed** and **awk** scripts into Perl scripts.

But wait, there's more...

Perl version 5 is nearly a complete rewrite, and provides the following additional benefits:

- **Many usability enhancements**

It is now possible to write much more readable Perl code (even within regular expressions). Formerly cryptic variable names can be replaced by mnemonic identifiers. Error messages are more informative, and the optional warnings will catch many of the mistakes a novice might make. This cannot be stressed enough. Whenever you get mysterious behavior, try the **-w** switch!!! Whenever you don't get mysterious behavior, try using **-w** anyway.

- **Simplified grammar**

The new yacc grammar is one half the size of the old one. Many of the arbitrary grammar rules have been regularized. The number of reserved words has been cut by 2/3. Despite this, nearly all old Perl scripts will continue to work unchanged.

- **Lexical scoping**

Perl variables may now be declared within a lexical scope, like "auto" variables in C. Not only is this more efficient, but it contributes to better privacy for "programming in the large". Anonymous subroutines exhibit deep binding of lexical variables (closures).

- **Arbitrarily nested data structures**

Any scalar value, including any array element, may now contain a reference to any other variable or subroutine. You can easily create anonymous variables and subroutines. Perl manages your reference counts for you.

- **Modularity and reusability**

The Perl library is now defined in terms of modules which can be easily shared among various packages. A package may choose to import all or a portion of a module's published interface. Pragmas (that is, compiler directives) are defined and used by the same mechanism.

- **Object-oriented programming**

A package can function as a class. Dynamic multiple inheritance and virtual methods are supported in a straightforward manner and with very little new syntax. Filehandles may now be treated as objects.

- **Embeddable and Extensible**

Perl may now be embedded easily in your C or C++ application, and can either call or be called by your routines through a documented interface. The XS pre-processor is provided to make it easy to glue your C or C++ routines into Perl. Dynamic loading of modules is supported, and Perl itself can be made into a dynamic library.

- **POSIX compliant**

A major new module is the POSIX module, which provides access to all available POSIX routines and definitions, via object classes where appropriate.

- **Package constructors and destructors**

The new BEGIN and END blocks provide means to capture control as a package is being compiled, and after the program exits. As a degenerate case they work just like awk's BEGIN and END when you use the `-p` or `-n` switches.

- **Multiple simultaneous DBM implementations**

A Perl program may now access DBM, NDBM, SDBM, GDBM, and Berkeley DB files from the same script simultaneously. In fact, the old dbmopen interface has been generalized to allow any variable to be tied to an object class which defines its access methods.

- **Subroutine definitions may now be autoloaded**

In fact, the AUTOLOAD mechanism also allows you to define any arbitrary semantics for undefined subroutine calls. It's not for just autoloading.

- **Regular expression enhancements**

You can now specify nongreedy quantifiers. You can now do grouping without creating a backreference. You can now write regular expressions with embedded whitespace and comments for readability. A consistent extensibility mechanism has been added that is upwardly compatible with all old regular expressions.

- **Innumerable Unbundled Modules**

The Comprehensive Perl Archive Network described in the *perlmodlib* manpage contains hundreds of plug-and-play modules full of reusable code. See <http://www.perl.com/CPAN> for a site near you.

- **Compilability**

While not yet in full production mode, a working perl-to-C compiler does exist. It can generate portable byte code, simple C, or optimized C code.

Okay, that's *definitely* enough hype.

## ENVIRONMENT

See the *perlrun* manpage.

## AUTHOR

Larry Wall <[larry@wall.org](mailto:larry@wall.org)>, with the help of oodles of other folks.

If your Perl success stories and testimonials may be of help to others who wish to advocate the use of Perl in their applications, or if you wish to simply express your gratitude to Larry and the Perl developers, please write to <[perl-thanks@perl.org](mailto:perl-thanks@perl.org)>.

## FILES

```
"/tmp/perl-e$$"      temporary file for -e commands
"@INC"              locations of perl libraries
```

## SEE ALSO

a2p     awk to perl translator  
s2p     sed to perl translator

## DIAGNOSTICS

The `-w` switch produces some lovely diagnostics.

See the *perldiag* manpage for explanations of all Perl's diagnostics. The use `diagnostics` pragma automatically turns Perl's normally terse warnings and errors into these longer forms.

Compilation errors will tell you the line number of the error, with an indication of the next token or token type that was to be examined. (In the case of a script passed to Perl via `-e` switches, each `-e` is counted as one line.)

Setuid scripts have additional constraints that can produce error messages such as "Insecure dependency". See the *perlsec* manpage.

Did we mention that you should definitely consider using the `-w` switch?

## BUGS

The `-w` switch is not mandatory.

Perl is at the mercy of your machine's definitions of various operations such as type casting, *atof()*, and floating-point output with *sprintf()*.

If your stdio requires a seek or eof between reads and writes on a particular stream, so does Perl. (This doesn't apply to *sysread()* and *syswrite()*.)

While none of the built-in data types have any arbitrary size limits (apart from memory size), there are still a few arbitrary limits: a given variable name may not be longer than 255 characters, and no component of your PATH may be longer than 255 if you use `-S`. A regular expression may not compile to more than 32767 bytes internally.

You may mail your bug reports (be sure to include full configuration information as output by the *myconfig* program in the perl source tree, or by `perl -V`) to `<perl-bug@perl.com>`. If you've succeeded in compiling perl, the *perlbug* script in the *utils/* subdirectory can be used to help mail in a bug report.

Perl actually stands for Pathologically Eclectic Rubbish Lister, but don't tell anyone I said that.

## NOTES

The Perl motto is "There's more than one way to do it." Divining how many more is left as an exercise to the reader.

The three principal virtues of a programmer are Laziness, Impatience, and Hubris. See the Camel Book for why.

