NAME

perlglossary - Perl Glossary

DESCRIPTION

A glossary of terms (technical and otherwise) used in the Perl documentation. Other useful sources include the Free On-Line Dictionary of Computing *http://foldoc.doc.ic.ac.uk/foldoc/index.html*, the Jargon File *http://catb.org/~esr/jargon/*, and Wikipedia *http://www.wikipedia.org/*.

A

accessor methods

A *method* used to indirectly inspect or update an *object*'s state (its *instance variables*).

actual arguments

The *scalar values* that you supply to a *function* or *subroutine* when you call it. For instance, when you call `power("puff")`, the string `"puff"` is the actual argument. See also *argument* and *formal arguments*.

address operator

Some languages work directly with the memory addresses of values, but this can be like playing with fire. Perl provides a set of asbestos gloves for handling all memory management. The closest to an address operator in Perl is the backslash operator, but it gives you a *hard reference*, which is much safer than a memory address.

algorithm

A well-defined sequence of steps, clearly enough explained that even a computer could do them.

alias

A nickname for something, which behaves in all ways as though you'd used the original name instead of the nickname. Temporary aliases are implicitly created in the loop variable for `foreach` loops, in the `$_` variable for *map* or *grep* operators, in `$a` and `$b` during *sort*'s comparison function, and in each element of `@_` for the *actual arguments* of a subroutine call. Permanent aliases are explicitly created in *packages* by *importing* symbols or by assignment to *typeglobs*. Lexically scoped aliases for package variables are explicitly created by the *our* declaration.

alternatives

A list of possible choices from which you may select only one, as in "Would you like door A, B, or C?" Alternatives in regular expressions are separated with a single vertical bar: `|`. Alternatives in normal Perl expressions are separated with a double vertical bar: `||`. Logical alternatives in *Boolean* expressions are separated with either `||` or `or`.

anonymous

Used to describe a *referent* that is not directly accessible through a named *variable*. Such a referent must be indirectly accessible through at least one *hard reference*. When the last hard reference goes away, the anonymous referent is destroyed without pity.

architecture

The kind of computer you're working on, where one "kind" of computer means all those computers sharing a compatible machine language. Since Perl programs are (typically) simple text files, not executable images, a Perl program is much less sensitive to the architecture it's running on than programs in other languages, such as C, that are compiled into machine code. See also *platform* and *operating system*.

argument

A piece of data supplied to a *program*, *subroutine*, *function*, or *method* to tell it what it's supposed to do. Also called a "parameter".

ARGV

The name of the array containing the *argument vector* from the command line. If you use the empty <> operator, *ARGV* is the name of both the *filehandle* used to traverse the arguments and the *scalar* containing the name of the current input file.

arithmetical operator

A *symbol* such as + or / that tells Perl to do the arithmetic you were supposed to learn in grade school.

array

An ordered sequence of *values*, stored such that you can easily access any of the values using an integer *subscript* that specifies the value's *offset* in the sequence.

array context

An archaic expression for what is more correctly referred to as *list context*.

ASCII

The American Standard Code for Information Interchange (a 7-bit character set adequate only for poorly representing English text). Often used loosely to describe the lowest 128 values of the various ISO-8859-X character sets, a bunch of mutually incompatible 8-bit codes sometimes described as half ASCII. See also *Unicode*.

assertion

A component of a *regular expression* that must be true for the pattern to match but does not necessarily match any characters itself. Often used specifically to mean a *zero width* assertion.

assignment

An *operator* whose assigned mission in life is to change the value of a *variable*.

assignment operator

Either a regular *assignment*, or a compound *operator* composed of an ordinary assignment and some other operator, that changes the value of a variable in place, that is, relative to its old value. For example, $a += 2 adds 2 to $a.

associative array

See *hash*. Please.

associativity

Determines whether you do the left *operator* first or the right *operator* first when you have "A *operator* B *operator* C" and the two operators are of the same precedence. Operators like + are left associative, while operators like ** are right associative. See *perlop* for a list of operators and their associativity.

asynchronous

Said of events or activities whose relative temporal ordering is indeterminate because too many things are going on at once. Hence, an asynchronous event is one you didn't know when to expect.

atom

A *regular expression* component potentially matching a *substring* containing one or more characters and treated as an indivisible syntactic unit by any following *quantifier*. (Contrast with an *assertion* that matches something of *zero width* and may not be quantified.)

atomic operation

When Democritus gave the word "atom" to the indivisible bits of matter, he meant literally something that could not be cut: *a-* (not) + *tomos* (cuttable). An atomic operation is an action that can't be interrupted, not one forbidden in a nuclear-free zone.

attribute

A new feature that allows the declaration of *variables* and *subroutines* with modifiers as in `sub foo : locked method`. Also, another name for an *instance variable* of an *object*.

autogeneration

A feature of *operator overloading* of *objects*, whereby the behavior of certain *operators* can be reasonably deduced using more fundamental operators. This assumes that the overloaded operators will often have the same relationships as the regular operators. See *perlop*.

autoincrement

To add one to something automatically, hence the name of the ++ operator. To instead subtract one from something automatically is known as an "autodecrement".

autoload

To load on demand. (Also called "lazy" loading.) Specifically, to call an *AUTOLOAD* subroutine on behalf of an undefined subroutine.

autosplit

To split a string automatically, as the **-a** *switch* does when running under **-p** or **-n** in order to emulate *awk*. (See also the *AutoSplit* module, which has nothing to do with the **-a** switch, but a lot to do with autoloading.)

autovivification

A Greco-Roman word meaning "to bring oneself to life". In Perl, storage locations (*lvalues*) spontaneously generate themselves as needed, including the creation of any *hard reference* values to point to the next level of storage. The assignment `$a[5][5][5][5][5] = "quintet"` potentially creates five scalar storage locations, plus four references (in the first four scalar locations) pointing to four new anonymous arrays (to hold the last four scalar locations). But the point of autovivification is that you don't have to worry about it.

AV

Short for "array value", which refers to one of Perl's internal data types that holds an *array*. The *AV* type is a subclass of *SV*.

awk

Descriptive editing term--short for "awkward". Also coincidentally refers to a venerable text-processing language from which Perl derived some of its high-level ideas.

B

backreference

A substring *captured* by a subpattern within unadorned parentheses in a *regex*, also referred to as a capture group. The sequences (`\g1`, `\g2`, etc.) later in the same pattern refer back to the corresponding subpattern in the current match. Outside the pattern, the numbered variables (`$1`, `$2`, etc.) continue to refer to these same values, as long as the pattern was the last successful match of the current dynamic scope. `\g{-1}` can be used to refer to a group by relative rather than absolute position; and groups can be also be named, and referred to later by name rather than number. See *"Capture groups" in perlre*.

backtracking

The practice of saying, "If I had to do it all over, I'd do it differently," and then actually going

back and doing it all over differently. Mathematically speaking, it's returning from an unsuccessful recursion on a tree of possibilities. Perl backtracks when it attempts to match patterns with a *regular expression*, and its earlier attempts don't pan out. See *"Backtracking" in perlre*.

backward compatibility

Means you can still run your old program because we didn't break any of the features or bugs it was relying on.

bareword

A word sufficiently ambiguous to be deemed illegal under *use strict 'subs'*. In the absence of that stricture, a bareword is treated as if quotes were around it.

base class

A generic *object* type; that is, a *class* from which other, more specific classes are derived genetically by *inheritance*. Also called a "superclass" by people who respect their ancestors.

big-endian

From Swift: someone who eats eggs big end first. Also used of computers that store the most significant *byte* of a word at a lower byte address than the least significant byte. Often considered superior to little-endian machines. See also *little-endian*.

binary

Having to do with numbers represented in base 2. That means there's basically two numbers, 0 and 1. Also used to describe a "non-text file", presumably because such a file makes full use of all the binary bits in its bytes. With the advent of *Unicode*, this distinction, already suspect, loses even more of its meaning.

binary operator

An *operator* that takes two *operands*.

bind

To assign a specific *network address* to a *socket*.

bit

An integer in the range from 0 to 1, inclusive. The smallest possible unit of information storage. An eighth of a *byte* or of a dollar. (The term "Pieces of Eight" comes from being able to split the old Spanish dollar into 8 bits, each of which still counted for money. That's why a 25-cent piece today is still "two bits".)

bit shift

The movement of bits left or right in a computer word, which has the effect of multiplying or dividing by a power of 2.

bit string

A sequence of *bits* that is actually being thought of as a sequence of bits, for once.

bless

In corporate life, to grant official approval to a thing, as in, "The VP of Engineering has blessed our WebCruncher project." Similarly in Perl, to grant official approval to a *referent* so that it can function as an *object*, such as a WebCruncher object. See *"bless" in perlfunc*.

block

What a *process* does when it has to wait for something: "My process blocked waiting for the disk." As an unrelated noun, it refers to a large chunk of data, of a size that the *operating system* likes to deal with (normally a power of two such as 512 or 8192). Typically refers to a

chunk of data that's coming from or going to a disk file.

BLOCK

A syntactic construct consisting of a sequence of Perl *statements* that is delimited by braces. The `if` and `while` statements are defined in terms of *BLOCKs*, for instance. Sometimes we also say "block" to mean a lexical scope; that is, a sequence of statements that act like a *BLOCK*, such as within an *eval* or a file, even though the statements aren't delimited by braces.

block buffering

A method of making input and output efficient by passing one *block* at a time. By default, Perl does block buffering to disk files. See *buffer* and *command buffering*.

Boolean

A value that is either *true* or *false*.

Boolean context

A special kind of *scalar context* used in conditionals to decide whether the *scalar value* returned by an expression is *true* or *false*. Does not evaluate as either a string or a number. See *context*.

breakpoint

A spot in your program where you've told the debugger to stop *execution* so you can poke around and see whether anything is wrong yet.

broadcast

To send a *datagram* to multiple destinations simultaneously.

BSD

A psychoactive drug, popular in the 80s, probably developed at U. C. Berkeley or thereabouts. Similar in many ways to the prescription-only medication called "System V", but infinitely more useful. (Or, at least, more fun.) The full chemical name is "Berkeley Standard Distribution".

bucket

A location in a *hash table* containing (potentially) multiple entries whose keys "hash" to the same hash value according to its hash function. (As internal policy, you don't have to worry about it, unless you're into internals, or policy.)

buffer

A temporary holding location for data. *Block buffering* means that the data is passed on to its destination whenever the buffer is full. *Line buffering* means that it's passed on whenever a complete line is received. *Command buffering* means that it's passed every time you do a *print* command (or equivalent). If your output is unbuffered, the system processes it one byte at a time without the use of a holding area. This can be rather inefficient.

built-in

A *function* that is predefined in the language. Even when hidden by *overriding*, you can always get at a built-in function by *qualifying* its name with the `CORE::` pseudo-package.

bundle

A group of related modules on *CPAN*. (Also, sometimes refers to a group of command-line switches grouped into one *switch cluster*.)

byte

A piece of data worth eight *bits* in most places.

bytecode

A pidgin-like language spoken among 'droids when they don't wish to reveal their orientation (see *endian*). Named after some similar languages spoken (for similar reasons) between compilers and interpreters in the late 20th century. These languages are characterized by representing everything as a non-architecture-dependent sequence of bytes.

C

C

A language beloved by many for its inside-out *type* definitions, inscrutable *precedence* rules, and heavy *overloading* of the function-call mechanism. (Well, actually, people first switched to C because they found lowercase identifiers easier to read than upper.) Perl is written in C, so it's not surprising that Perl borrowed a few ideas from it.

C preprocessor

The typical C compiler's first pass, which processes lines beginning with # for conditional compilation and macro definition and does various manipulations of the program text based on the current definitions. Also known as *cpp*(1).

call by reference

An *argument*-passing mechanism in which the *formal arguments* refer directly to the *actual arguments*, and the *subroutine* can change the actual arguments by changing the formal arguments. That is, the formal argument is an *alias* for the actual argument. See also *call by value*.

call by value

An *argument*-passing mechanism in which the *formal arguments* refer to a copy of the *actual arguments*, and the *subroutine* cannot change the actual arguments by changing the formal arguments. See also *call by reference*.

callback

A *handler* that you register with some other part of your program in the hope that the other part of your program will *trigger* your handler when some event of interest transpires.

canonical

Reduced to a standard form to facilitate comparison.

capture buffer, capture group

These two terms are synonymous: a *captured substring* by a regex subpattern.

capturing

The use of parentheses around a *subpattern* in a *regular expression* to store the matched *substring* as a *backreference* or *capture group*. (Captured strings are also returned as a list in *list context*.)

character

A small integer representative of a unit of orthography. Historically, characters were usually stored as fixed-width integers (typically in a byte, or maybe two, depending on the character set), but with the advent of UTF-8, characters are often stored in a variable number of bytes depending on the size of the integer that represents the character. Perl manages this transparently for you, for the most part.

character class

A square-bracketed list of characters used in a *regular expression* to indicate that any character of the set may occur at a given point. Loosely, any predefined set of characters so used.

character property

A predefined *character class* matchable by the `\p` *metasymbol*. Many standard properties are defined for *Unicode*.

circumfix operator

An *operator* that surrounds its *operand*, like the angle operator, or parentheses, or a hug.

class

A user-defined *type*, implemented in Perl via a *package* that provides (either directly or by inheritance) *methods* (that is, *subroutines*) to handle *instances* of the class (its *objects*). See also *inheritance*.

class method

A *method* whose *invocand* is a *package* name, not an *object* reference. A method associated with the class as a whole.

client

In networking, a *process* that initiates contact with a *server* process in order to exchange data and perhaps receive a service.

cloister

A *cluster* used to restrict the scope of a *regular expression modifier*.

closure

An *anonymous* subroutine that, when a reference to it is generated at run time, keeps track of the identities of externally visible *lexical variables* even after those lexical variables have supposedly gone out of *scope*. They're called "closures" because this sort of behavior gives mathematicians a sense of closure.

cluster

A parenthesized *subpattern* used to group parts of a *regular expression* into a single *atom*.

CODE

The word returned by the *ref* function when you apply it to a reference to a subroutine. See also *CV*.

code generator

A system that writes code for you in a low-level language, such as code to implement the backend of a compiler. See *program generator*.

code point

The position of a character in a character set encoding. The character `NULL` is almost certainly at the zeroth position in all character sets, so its code point is 0. The code point for the `SPACE` character in the ASCII character set is 0x20, or 32 decimal; in EBCDIC it is 0x40, or 64 decimal. The *ord* function returns the code point of a character.

"code position" and "ordinal" mean the same thing as "code point".

code subpattern

A *regular expression* subpattern whose real purpose is to execute some Perl code, for example, the `(?{...})` and `(??{...})` subpatterns.

collating sequence

The order into which *characters* sort. This is used by *string* comparison routines to decide, for example, where in this glossary to put "collating sequence".

command

In *shell* programming, the syntactic combination of a program name and its arguments. More loosely, anything you type to a shell (a command interpreter) that starts it doing something. Even more loosely, a Perl *statement*, which might start with a *label* and typically ends with a semicolon.

command buffering

A mechanism in Perl that lets you store up the output of each Perl *command* and then flush it out as a single request to the *operating system*. It's enabled by setting the `$|` (`$AUTOFLUSH`) variable to a true value. It's used when you don't want data sitting around not going where it's supposed to, which may happen because the default on a *file* or *pipe* is to use *block buffering*.

command name

The name of the program currently executing, as typed on the command line. In C, the *command* name is passed to the program as the first command-line argument. In Perl, it comes in separately as `$0`.

command-line arguments

The *values* you supply along with a program name when you tell a *shell* to execute a *command*. These values are passed to a Perl program through `@ARGV`.

comment

A remark that doesn't affect the meaning of the program. In Perl, a comment is introduced by a # character and continues to the end of the line.

compilation unit

The *file* (or *string*, in the case of *eval*) that is currently being compiled.

compile phase

Any time before Perl starts running your main program. See also *run phase*. Compile phase is mostly spent in *compile time*, but may also be spent in *run time* when `BEGIN` blocks, *use* declarations, or constant subexpressions are being evaluated. The startup and import code of any *use* declaration is also run during compile phase.

compile time

The time when Perl is trying to make sense of your code, as opposed to when it thinks it knows what your code means and is merely trying to do what it thinks your code says to do, which is *run time*.

compiler

Strictly speaking, a program that munches up another program and spits out yet another file containing the program in a "more executable" form, typically containing native machine instructions. The *perl* program is not a compiler by this definition, but it does contain a kind of compiler that takes a program and turns it into a more executable form (*syntax trees*) within the *perl* process itself, which the *interpreter* then interprets. There are, however, extension *modules* to get Perl to act more like a "real" compiler. See *O*.

composer

A "constructor" for a *referent* that isn't really an *object*, like an anonymous array or a hash (or a sonata, for that matter). For example, a pair of braces acts as a composer for a hash, and a pair of brackets acts as a composer for an array. See *"Making References" in perlref*.

concatenation

The process of gluing one cat's nose to another cat's tail. Also, a similar operation on two *strings*.

conditional

Something "iffy". See *Boolean context*.

connection

In telephony, the temporary electrical circuit between the caller's and the callee's phone. In networking, the same kind of temporary circuit between a *client* and a *server*.

construct

As a noun, a piece of syntax made up of smaller pieces. As a transitive verb, to create an *object* using a *constructor*.

constructor

Any *class method*, instance *method*, or *subroutine* that composes, initializes, blesses, and returns an *object*. Sometimes we use the term loosely to mean a *composer*.

context

The surroundings, or environment. The context given by the surrounding code determines what kind of data a particular *expression* is expected to return. The three primary contexts are *list context*, *scalar context*, and *void context*. Scalar context is sometimes subdivided into *Boolean context*, *numeric context*, *string context*, and *void context*. There's also a "don't care" scalar context (which is dealt with in Programming Perl, Third Edition, Chapter 2, "Bits and Pieces" if you care).

continuation

The treatment of more than one physical *line* as a single logical line. *Makefile* lines are continued by putting a backslash before the *newline*. Mail headers as defined by RFC 822 are continued by putting a space or tab *after* the newline. In general, lines in Perl do not need any form of continuation mark, because *whitespace* (including newlines) is gleefully ignored. Usually.

core dump

The corpse of a *process*, in the form of a file left in the *working directory* of the process, usually as a result of certain kinds of fatal error.

CPAN

The Comprehensive Perl Archive Network. (See *"What modules and extensions are available for Perl? What is CPAN? What does CPAN/src/... mean?" in perlfaq2*).

cracker

Someone who breaks security on computer systems. A cracker may be a true *hacker* or only a *script kiddie*.

current package

The *package* in which the current statement is compiled. Scan backwards in the text of your program through the current *lexical scope* or any enclosing lexical scopes till you find a package declaration. That's your current package name.

current working directory

See *working directory*.

currently selected output channel

The last *filehandle* that was designated with *select*(`FILEHANDLE`); *STDOUT*, if no filehandle has been selected.

CV

An internal "code value" typedef, holding a *subroutine*. The *CV* type is a subclass of *SV*.

D

dangling statement

A bare, single *statement*, without any braces, hanging off an `if` or `while` conditional. C allows them. Perl doesn't.

data structure

How your various pieces of data relate to each other and what shape they make when you put them all together, as in a rectangular table or a triangular-shaped tree.

data type

A set of possible values, together with all the operations that know how to deal with those values. For example, a numeric data type has a certain set of numbers that you can work with and various mathematical operations that you can do on the numbers but would make little sense on, say, a string such as `"Kilroy"`. Strings have their own operations, such as *concatenation*. Compound types made of a number of smaller pieces generally have operations to compose and decompose them, and perhaps to rearrange them. *Objects* that model things in the real world often have operations that correspond to real activities. For instance, if you model an elevator, your elevator object might have an `open_door()` *method*.

datagram

A packet of data, such as a *UDP* message, that (from the viewpoint of the programs involved) can be sent independently over the network. (In fact, all packets are sent independently at the *IP* level, but *stream* protocols such as *TCP* hide this from your program.)

DBM

Stands for "Data Base Management" routines, a set of routines that emulate an *associative array* using disk files. The routines use a dynamic hashing scheme to locate any entry with only two disk accesses. DBM files allow a Perl program to keep a persistent *hash* across multiple invocations. You can *tie* your hash variables to various DBM implementations--see *AnyDBM_File* and *DB_File*.

declaration

An *assertion* that states something exists and perhaps describes what it's like, without giving any commitment as to how or where you'll use it. A declaration is like the part of your recipe that says, "two cups flour, one large egg, four or five tadpoles..." See *statement* for its opposite. Note that some declarations also function as statements. Subroutine declarations also act as definitions if a body is supplied.

decrement

To subtract a value from a variable, as in "decrement $x" (meaning to remove 1 from its value) or "decrement $x by 3".

default

A *value* chosen for you if you don't supply a value of your own.

defined

Having a meaning. Perl thinks that some of the things people try to do are devoid of meaning, in particular, making use of variables that have never been given a *value* and performing certain operations on data that isn't there. For example, if you try to read data past the end of a file, Perl will hand you back an undefined value. See also *false* and *"defined" in perlfunc*.

delimiter

A *character* or *string* that sets bounds to an arbitrarily-sized textual object, not to be confused with a *separator* or *terminator*. "To delimit" really just means "to surround" or "to enclose" (like these parentheses are doing).

deprecated modules and features

Deprecated modules and features are those which were part of a stable release, but later found to be subtly flawed, and which should be avoided. They are subject to removal and/or bug-incompatible reimplementation in the next major release (but they will be preserved through maintenance releases). Deprecation warnings are issued under **-w** or `use diagnostics`, and notices are found in *perldelta*s, as well as various other PODs. Coding practices that misuse features, such as `my $foo if 0`, can also be deprecated.

dereference

A fancy computer science term meaning "to follow a *reference* to what it points to". The "de" part of it refers to the fact that you're taking away one level of *indirection*.

derived class

A *class* that defines some of its *methods* in terms of a more generic class, called a *base class*. Note that classes aren't classified exclusively into base classes or derived classes: a class can function as both a derived class and a base class simultaneously, which is kind of classy.

descriptor

See *file descriptor*.

destroy

To deallocate the memory of a *referent* (first triggering its `DESTROY` method, if it has one).

destructor

A special *method* that is called when an *object* is thinking about *destroying* itself. A Perl program's `DESTROY` method doesn't do the actual destruction; Perl just *triggers* the method in case the *class* wants to do any associated cleanup.

device

A whiz-bang hardware gizmo (like a disk or tape drive or a modem or a joystick or a mouse) attached to your computer, that the *operating system* tries to make look like a *file* (or a bunch of files). Under Unix, these fake files tend to live in the */dev* directory.

directive

A *pod* directive. See *perlpod*.

directory

A special file that contains other files. Some *operating systems* call these "folders", "drawers", or "catalogs".

directory handle

A name that represents a particular instance of opening a directory to read it, until you close it. See the *opendir* function.

dispatch

To send something to its correct destination. Often used metaphorically to indicate a transfer of programmatic control to a destination selected algorithmically, often by lookup in a table of function *references* or, in the case of object *methods*, by traversing the inheritance tree looking for the most specific definition for the method.

distribution

A standard, bundled release of a system of software. The default usage implies source code is included. If that is not the case, it will be called a "binary-only" distribution.

(to be) dropped modules

When Perl 5 was first released (see *perlhist*), several modules were included, which have now

fallen out of common use. It has been suggested that these modules should be removed, since the distribution became rather large, and the common criterion for new module additions is now limited to modules that help to build, test, and extend perl itself. Furthermore, the CPAN (which didn't exist at the time of Perl 5.0) can become the new home of dropped modules. Dropping modules is currently not an option, but further developments may clear the last barriers.

dweomer

An enchantment, illusion, phantasm, or jugglery. Said when Perl's magical *dwimmer* effects don't do what you expect, but rather seem to be the product of arcane dweomercraft, sorcery, or wonder working. [From Old English]

dwimmer

DWIM is an acronym for "Do What I Mean", the principle that something should just do what you want it to do without an undue amount of fuss. A bit of code that does "dwimming" is a "dwimmer". Dwimming can require a great deal of behind-the-scenes magic, which (if it doesn't stay properly behind the scenes) is called a *dweomer* instead.

dynamic scoping

Dynamic scoping works over a dynamic scope, making variables visible throughout the rest of the *block* in which they are first used and in any *subroutines* that are called by the rest of the block. Dynamically scoped variables can have their values temporarily changed (and implicitly restored later) by a *local* operator. (Compare *lexical scoping*.) Used more loosely to mean how a subroutine that is in the middle of calling another subroutine "contains" that subroutine at *run time*.

E

eclectic

Derived from many sources. Some would say *too* many.

element

A basic building block. When you're talking about an *array*, it's one of the items that make up the array.

embedding

When something is contained in something else, particularly when that might be considered surprising: "I've embedded a complete Perl interpreter in my editor!"

empty subclass test

The notion that an empty *derived class* should behave exactly like its *base class*.

en passant

When you change a *value* as it is being copied. [From French, "in passing", as in the exotic pawn-capturing maneuver in chess.]

encapsulation

The veil of abstraction separating the *interface* from the *implementation* (whether enforced or not), which mandates that all access to an *object*'s state be through *methods* alone.

endian

See *little-endian* and *big-endian*.

environment

The collective set of *environment variables* your *process* inherits from its parent. Accessed via %ENV.

environment variable

A mechanism by which some high-level agent such as a user can pass its preferences down to its future offspring (child *processes*, grandchild processes, great-grandchild processes, and so on). Each environment variable is a *key*/*value* pair, like one entry in a *hash*.

EOF

End of File. Sometimes used metaphorically as the terminating string of a *here document*.

errno

The error number returned by a *syscall* when it fails. Perl refers to the error by the name `$!` (or `$OS_ERROR` if you use the English module).

error

See *exception* or *fatal error*.

escape sequence

See *metasymbol*.

exception

A fancy term for an error. See *fatal error*.

exception handling

The way a program responds to an error. The exception handling mechanism in Perl is the *eval* operator.

exec

To throw away the current *process*'s program and replace it with another without exiting the process or relinquishing any resources held (apart from the old memory image).

executable file

A *file* that is specially marked to tell the *operating system* that it's okay to run this file as a program. Usually shortened to "executable".

execute

To run a *program* or *subroutine*. (Has nothing to do with the *kill* built-in, unless you're trying to run a *signal handler*.)

execute bit

The special mark that tells the operating system it can run this program. There are actually three execute bits under Unix, and which bit gets used depends on whether you own the file singularly, collectively, or not at all.

exit status

See *status*.

export

To make symbols from a *module* available for *import* by other modules.

expression

Anything you can legally say in a spot where a *value* is required. Typically composed of *literals*, *variables*, *operators*, *functions*, and *subroutine* calls, not necessarily in that order.

extension

A Perl module that also pulls in compiled C or C++ code. More generally, any experimental option that can be compiled into Perl, such as multithreading.

F

false

In Perl, any value that would look like `""` or `"0"` if evaluated in a string context. Since undefined values evaluate to `""`, all undefined values are false, but not all false values are undefined.

FAQ

Frequently Asked Question (although not necessarily frequently answered, especially if the answer appears in the Perl FAQ shipped standard with Perl).

fatal error

An uncaught *exception*, which causes termination of the *process* after printing a message on your *standard error* stream. Errors that happen inside an *eval* are not fatal. Instead, the *eval* terminates after placing the exception message in the $@ (`$EVAL_ERROR`) variable. You can try to provoke a fatal error with the *die* operator (known as throwing or raising an exception), but this may be caught by a dynamically enclosing *eval*. If not caught, the *die* becomes a fatal error.

field

A single piece of numeric or string data that is part of a longer *string*, *record*, or *line*. Variable-width fields are usually split up by *separators* (so use *split* to extract the fields), while fixed-width fields are usually at fixed positions (so use *unpack*). *Instance variables* are also known as fields.

FIFO

First In, First Out. See also *LIFO*. Also, a nickname for a *named pipe*.

file

A named collection of data, usually stored on disk in a *directory* in a *filesystem*. Roughly like a document, if you're into office metaphors. In modern filesystems, you can actually give a file more than one name. Some files have special properties, like directories and devices.

file descriptor

The little number the *operating system* uses to keep track of which opened *file* you're talking about. Perl hides the file descriptor inside a *standard I/O* stream and then attaches the stream to a *filehandle*.

file test operator

A built-in unary operator that you use to determine whether something is *true* about a file, such as `-o $filename` to test whether you're the owner of the file.

fileglob

A "wildcard" match on *filenames*. See the *glob* function.

filehandle

An identifier (not necessarily related to the real name of a file) that represents a particular instance of opening a file until you close it. If you're going to open and close several different files in succession, it's fine to open each of them with the same filehandle, so you don't have to write out separate code to process each file.

filename

One name for a file. This name is listed in a *directory*, and you can use it in an *open* to tell the *operating system* exactly which file you want to open, and associate the file with a *filehandle* which will carry the subsequent identity of that file in your program, until you close it.

filesystem

A set of *directories* and *files* residing on a partition of the disk. Sometimes known as a "partition". You can change the file's name or even move a file around from directory to directory within a filesystem without actually moving the file itself, at least under Unix.

filter

A program designed to take a *stream* of input and transform it into a stream of output.

flag

We tend to avoid this term because it means so many things. It may mean a command-line *switch* that takes no argument itself (such as Perl's **-n** and **-p** flags) or, less frequently, a single-bit indicator (such as the `O_CREAT` and `O_EXCL` flags used in *sysopen*).

floating point

A method of storing numbers in "scientific notation", such that the precision of the number is independent of its magnitude (the decimal point "floats"). Perl does its numeric work with floating-point numbers (sometimes called "floats"), when it can't get away with using *integers*. Floating-point numbers are mere approximations of real numbers.

flush

The act of emptying a *buffer*, often before it's full.

FMTEYEWTK

Far More Than Everything You Ever Wanted To Know. An exhaustive treatise on one narrow topic, something of a super-*FAQ*. See Tom for far more.

fork

To create a child *process* identical to the parent process at its moment of conception, at least until it gets ideas of its own. A thread with protected memory.

formal arguments

The generic names by which a *subroutine* knows its *arguments*. In many languages, formal arguments are always given individual names, but in Perl, the formal arguments are just the elements of an array. The formal arguments to a Perl program are `$ARGV[0]`, `$ARGV[1]`, and so on. Similarly, the formal arguments to a Perl subroutine are `$_[0]`, `$_[1]`, and so on. You may give the arguments individual names by assigning the values to a *my* list. See also *actual arguments*.

format

A specification of how many spaces and digits and things to put somewhere so that whatever you're printing comes out nice and pretty.

freely available

Means you don't have to pay money to get it, but the copyright on it may still belong to someone else (like Larry).

freely redistributable

Means you're not in legal trouble if you give a bootleg copy of it to your friends and we find out about it. In fact, we'd rather you gave a copy to all your friends.

freeware

Historically, any software that you give away, particularly if you make the source code available as well. Now often called `open source software`. Recently there has been a trend to use the term in contradistinction to *open source software*, to refer only to free software released under the Free Software Foundation's GPL (General Public License), but this is difficult to justify etymologically.

function

Mathematically, a mapping of each of a set of input values to a particular output value. In computers, refers to a *subroutine* or *operator* that returns a *value*. It may or may not have input values (called *arguments*).

funny character

Someone like Larry, or one of his peculiar friends. Also refers to the strange prefixes that Perl requires as noun markers on its variables.

garbage collection

A misnamed feature--it should be called, "expecting your mother to pick up after you". Strictly speaking, Perl doesn't do this, but it relies on a reference-counting mechanism to keep things tidy. However, we rarely speak strictly and will often refer to the reference-counting scheme as a form of garbage collection. (If it's any comfort, when your interpreter exits, a "real" garbage collector runs to make sure everything is cleaned up if you've been messy with circular references and such.)

G

GID

Group ID--in Unix, the numeric group ID that the *operating system* uses to identify you and members of your *group*.

glob

Strictly, the shell's * character, which will match a "glob" of characters when you're trying to generate a list of filenames. Loosely, the act of using globs and similar symbols to do pattern matching. See also *fileglob* and *typeglob*.

global

Something you can see from anywhere, usually used of *variables* and *subroutines* that are visible everywhere in your program. In Perl, only certain special variables are truly global--most variables (and all subroutines) exist only in the current *package*. Global variables can be declared with *our*. See *"our" in perlfunc*.

global destruction

The *garbage collection* of globals (and the running of any associated object destructors) that takes place when a Perl *interpreter* is being shut down. Global destruction should not be confused with the Apocalypse, except perhaps when it should.

glue language

A language such as Perl that is good at hooking things together that weren't intended to be hooked together.

granularity

The size of the pieces you're dealing with, mentally speaking.

greedy

A *subpattern* whose *quantifier* wants to match as many things as possible.

grep

Originally from the old Unix editor command for "Globally search for a Regular Expression and Print it", now used in the general sense of any kind of search, especially text searches. Perl has a built-in *grep* function that searches a list for elements matching any given criterion, whereas the *grep*(1) program searches for lines matching a *regular expression* in one or more files.

group

A set of users of which you are a member. In some operating systems (like Unix), you can give certain file access permissions to other members of your group.

GV

An internal "glob value" typedef, holding a *typeglob*. The *GV* type is a subclass of *SV*.

H

hacker

Someone who is brilliantly persistent in solving technical problems, whether these involve golfing, fighting orcs, or programming. Hacker is a neutral term, morally speaking. Good hackers are not to be confused with evil *crackers* or clueless *script kiddies*. If you confuse them, we will presume that you are either evil or clueless.

handler

A *subroutine* or *method* that is called by Perl when your program needs to respond to some internal event, such as a *signal*, or an encounter with an operator subject to *operator overloading*. See also *callback*.

hard reference

A *scalar value* containing the actual address of a *referent*, such that the referent's *reference* count accounts for it. (Some hard references are held internally, such as the implicit reference from one of a *typeglob*'s variable slots to its corresponding referent.) A hard reference is different from a *symbolic reference*.

hash

An unordered association of *key*/*value* pairs, stored such that you can easily use a string *key* to look up its associated data *value*. This glossary is like a hash, where the word to be defined is the key, and the definition is the value. A hash is also sometimes septisyllabically called an "associative array", which is a pretty good reason for simply calling it a "hash" instead.

hash table

A data structure used internally by Perl for implementing associative arrays (hashes) efficiently. See also *bucket*.

header file

A file containing certain required definitions that you must include "ahead" of the rest of your program to do certain obscure operations. A C header file has a *.h* extension. Perl doesn't really have header files, though historically Perl has sometimes used translated *.h* files with a *.ph* extension. See *"require" in perlfunc*. (Header files have been superseded by the *module* mechanism.)

here document

So called because of a similar construct in *shells* that pretends that the *lines* following the *command* are a separate *file* to be fed to the command, up to some terminating string. In Perl, however, it's just a fancy form of quoting.

hexadecimal

A number in base 16, "hex" for short. The digits for 10 through 16 are customarily represented by the letters a through `f`. Hexadecimal constants in Perl start with `0x`. See also *"hex" in perlfunc*.

home directory

The directory you are put into when you log in. On a Unix system, the name is often placed into `$ENV{HOME}` or `$ENV{LOGDIR}` by *login*, but you can also find it with `(getpwuid($< ))[7]`. (Some platforms do not have a concept of a home directory.)

host

The computer on which a program or other data resides.

hubris

Excessive pride, the sort of thing Zeus zaps you for. Also the quality that makes you write (and maintain) programs that other people won't want to say bad things about. Hence, the third great virtue of a programmer. See also *laziness* and *impatience*.

HV

Short for a "hash value" typedef, which holds Perl's internal representation of a hash. The *HV* type is a subclass of *SV*.

I

identifier

A legally formed name for most anything in which a computer program might be interested. Many languages (including Perl) allow identifiers that start with a letter and contain letters and digits. Perl also counts the underscore character as a valid letter. (Perl also has more complicated names, such as *qualified* names.)

impatience

The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least that pretend to. Hence, the second great virtue of a programmer. See also *laziness* and *hubris*.

implementation

How a piece of code actually goes about doing its job. Users of the code should not count on implementation details staying the same unless they are part of the published *interface*.

import

To gain access to symbols that are exported from another module. See *"use" in perlfunc*.

increment

To increase the value of something by 1 (or by some other number, if so specified).

indexing

In olden days, the act of looking up a *key* in an actual index (such as a phone book), but now merely the act of using any kind of key or position to find the corresponding *value*, even if no index is involved. Things have degenerated to the point that Perl's *index* function merely locates the position (index) of one string in another.

indirect filehandle

An *expression* that evaluates to something that can be used as a *filehandle*: a *string* (filehandle name), a *typeglob*, a typeglob *reference*, or a low-level *IO* object.

indirect object

In English grammar, a short noun phrase between a verb and its direct object indicating the beneficiary or recipient of the action. In Perl, `print STDOUT "$foo\n";` can be understood as "verb indirect-object object" where *STDOUT* is the recipient of the *print* action, and `"$foo"` is the object being printed. Similarly, when invoking a *method*, you might place the invocand between the method and its arguments:

```
$gollum = new Pathetic::Creature "Smeagol";
give $gollum "Fisssssh!";
give $gollum "Precious!";
```

In modern Perl, calling methods this way is often considered bad practice and to be avoided.

indirect object slot

The syntactic position falling between a method call and its arguments when using the indirect object invocation syntax. (The slot is distinguished by the absence of a comma between it and the next argument.) *STDERR* is in the indirect object slot here:

```
print STDERR "Awake!  Awake!  Fear, Fire,
    Foes!  Awake!\n";
```

indirection

If something in a program isn't the value you're looking for but indicates where the value is, that's indirection. This can be done with either *symbolic references* or *hard references*.

infix

An *operator* that comes in between its *operands*, such as multiplication in `24 * 7`.

inheritance

What you get from your ancestors, genetically or otherwise. If you happen to be a *class*, your ancestors are called *base classes* and your descendants are called *derived classes*. See *single inheritance* and *multiple inheritance*.

instance

Short for "an instance of a class", meaning an *object* of that *class*.

instance variable

An *attribute* of an *object*; data stored with the particular object rather than with the class as a whole.

integer

A number with no fractional (decimal) part. A counting number, like 1, 2, 3, and so on, but including 0 and the negatives.

interface

The services a piece of code promises to provide forever, in contrast to its *implementation*, which it should feel free to change whenever it likes.

interpolation

The insertion of a scalar or list value somewhere in the middle of another value, such that it appears to have been there all along. In Perl, variable interpolation happens in double-quoted strings and patterns, and list interpolation occurs when constructing the list of values to pass to a list operator or other such construct that takes a *LIST*.

interpreter

Strictly speaking, a program that reads a second program and does what the second program says directly without turning the program into a different form first, which is what *compilers* do. Perl is not an interpreter by this definition, because it contains a kind of compiler that takes a program and turns it into a more executable form (*syntax trees*) within the *perl* process itself, which the Perl *run time* system then interprets.

invocand

The agent on whose behalf a *method* is invoked. In a *class* method, the invocand is a package name. In an *instance* method, the invocand is an object reference.

invocation

The act of calling up a deity, daemon, program, method, subroutine, or function to get it do what you think it's supposed to do. We usually "call" subroutines but "invoke" methods, since it sounds cooler.

I/O

Input from, or output to, a *file* or *device*.

IO

An internal I/O object. Can also mean *indirect object*.

IP

Internet Protocol, or Intellectual Property.

IPC

Interprocess Communication.

is-a

A relationship between two *objects* in which one object is considered to be a more specific version of the other, generic object: "A camel is a mammal." Since the generic object really only exists in a Platonic sense, we usually add a little abstraction to the notion of objects and think of the relationship as being between a generic *base class* and a specific *derived class*. Oddly enough, Platonic classes don't always have Platonic relationships--see *inheritance*.

iteration

Doing something repeatedly.

iterator

A special programming gizmo that keeps track of where you are in something that you're trying to iterate over. The `foreach` loop in Perl contains an iterator; so does a hash, allowing you to *each* through it.

IV

The integer four, not to be confused with six, Tom's favorite editor. IV also means an internal Integer Value of the type a *scalar* can hold, not to be confused with an *NV*.

J

JAPH

"Just Another Perl Hacker," a clever but cryptic bit of Perl code that when executed, evaluates to that string. Often used to illustrate a particular Perl feature, and something of an ongoing Obfuscated Perl Contest seen in Usenix signatures.

K

key

The string index to a *hash*, used to look up the *value* associated with that key.

keyword

See *reserved words*.

L

label

A name you give to a *statement* so that you can talk about that statement elsewhere in the program.

laziness

The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful, and document what you wrote so you don't have to answer so many questions about it. Hence, the first great virtue of a programmer. Also hence, this book. See also *impatience* and *hubris*.

left shift

A *bit shift* that multiplies the number by some power of 2.

leftmost longest

The preference of the *regular expression* engine to match the leftmost occurrence of a *pattern*, then given a position at which a match will occur, the preference for the longest match (presuming the use of a *greedy* quantifier). See *perlre* for *much* more on this subject.

lexeme

Fancy term for a *token*.

lexer

Fancy term for a *tokener*.

lexical analysis

Fancy term for *tokenizing*.

lexical scoping

Looking at your *Oxford English Dictionary* through a microscope. (Also known as *static scoping*, because dictionaries don't change very fast.) Similarly, looking at variables stored in a private dictionary (namespace) for each scope, which are visible only from their point of declaration down to the end of the lexical scope in which they are declared. --Syn. *static scoping*. --Ant. *dynamic scoping*.

lexical variable

A *variable* subject to *lexical scoping*, declared by *my*. Often just called a "lexical". (The *our* declaration declares a lexically scoped name for a global variable, which is not itself a lexical variable.)

library

Generally, a collection of procedures. In ancient days, referred to a collection of subroutines in a *.pl* file. In modern times, refers more often to the entire collection of Perl *modules* on your system.

LIFO

Last In, First Out. See also *FIFO*. A LIFO is usually called a *stack*.

line

In Unix, a sequence of zero or more non-newline characters terminated with a *newline* character. On non-Unix machines, this is emulated by the C library even if the underlying *operating system* has different ideas.

line buffering

Used by a *standard I/O* output stream that flushes its *buffer* after every *newline*. Many standard I/O libraries automatically set up line buffering on output that is going to the terminal.

line number

The number of lines read previous to this one, plus 1. Perl keeps a separate line number for each source or input file it opens. The current source file's line number is represented by `__LINE__`. The current input line number (for the file that was most recently read via <FH>) is represented by the `$.` (`$INPUT_LINE_NUMBER`) variable. Many error messages report both values, if available.

link

Used as a noun, a name in a *directory*, representing a *file*. A given file can have multiple links to it. It's like having the same phone number listed in the phone directory under different

names. As a verb, to resolve a partially compiled file's unresolved symbols into a (nearly) executable image. Linking can generally be static or dynamic, which has nothing to do with static or dynamic scoping.

LIST

A syntactic construct representing a comma-separated list of expressions, evaluated to produce a *list value*. Each *expression* in a *LIST* is evaluated in *list context* and interpolated into the list value.

list

An ordered set of scalar values.

list context

The situation in which an *expression* is expected by its surroundings (the code calling it) to return a list of values rather than a single value. Functions that want a *LIST* of arguments tell those arguments that they should produce a list value. See also *context*.

list operator

An *operator* that does something with a list of values, such as *join* or *grep*. Usually used for named built-in operators (such as *print*, *unlink*, and *system*) that do not require parentheses around their *argument* list.

list value

An unnamed list of temporary scalar values that may be passed around within a program from any list-generating function to any function or construct that provides a *list context*.

literal

A token in a programming language such as a number or *string* that gives you an actual *value* instead of merely representing possible values as a *variable* does.

little-endian

From Swift: someone who eats eggs little end first. Also used of computers that store the least significant *byte* of a word at a lower byte address than the most significant byte. Often considered superior to big-endian machines. See also *big-endian*.

local

Not meaning the same thing everywhere. A global variable in Perl can be localized inside a *dynamic scope* via the *local* operator.

logical operator

Symbols representing the concepts "and", "or", "xor", and "not".

lookahead

An *assertion* that peeks at the string to the right of the current match location.

lookbehind

An *assertion* that peeks at the string to the left of the current match location.

loop

A construct that performs something repeatedly, like a roller coaster.

loop control statement

Any statement within the body of a loop that can make a loop prematurely stop looping or skip an *iteration*. Generally you shouldn't try this on roller coasters.

loop label

A kind of key or name attached to a loop (or roller coaster) so that loop control statements can talk about which loop they want to control.

lvaluable

Able to serve as an *lvalue*.

lvalue

Term used by language lawyers for a storage location you can assign a new *value* to, such as a *variable* or an element of an *array*. The "l" is short for "left", as in the left side of an assignment, a typical place for lvalues. An *lvaluable* function or expression is one to which a value may be assigned, as in `pos($x) = 10`.

lvalue modifier

An adjectival pseudofunction that warps the meaning of an *lvalue* in some declarative fashion. Currently there are three lvalue modifiers: *my*, *our*, and *local*.

M

magic

Technically speaking, any extra semantics attached to a variable such as `$!`, `$0`, `%ENV`, or `%SIG`, or to any tied variable. Magical things happen when you diddle those variables.

magical increment

An *increment* operator that knows how to bump up alphabetics as well as numbers.

magical variables

Special variables that have side effects when you access them or assign to them. For example, in Perl, changing elements of the `%ENV` array also changes the corresponding environment variables that subprocesses will use. Reading the `$!` variable gives you the current system error number or message.

Makefile

A file that controls the compilation of a program. Perl programs don't usually need a *Makefile* because the Perl compiler has plenty of self-control.

man

The Unix program that displays online documentation (manual pages) for you.

manpage

A "page" from the manuals, typically accessed via the *man*(1) command. A manpage contains a SYNOPSIS, a DESCRIPTION, a list of BUGS, and so on, and is typically longer than a page. There are manpages documenting *commands*, *syscalls*, *library functions*, *devices*, *protocols*, *files*, and such. In this book, we call any piece of standard Perl documentation (like *perlop* or *perldelta*) a manpage, no matter what format it's installed in on your system.

matching

See *pattern matching*.

member data

See *instance variable*.

memory

This always means your main memory, not your disk. Clouding the issue is the fact that your machine may implement *virtual* memory; that is, it will pretend that it has more memory than it really does, and it'll use disk space to hold inactive bits. This can make it seem like you have a little more memory than you really do, but it's not a substitute for real memory. The best thing that can be said about virtual memory is that it lets your performance degrade gradually rather

than suddenly when you run out of real memory. But your program can die when you run out of virtual memory too, if you haven't thrashed your disk to death first.

metacharacter

A *character* that is *not* supposed to be treated normally. Which characters are to be treated specially as metacharacters varies greatly from context to context. Your *shell* will have certain metacharacters, double-quoted Perl *strings* have other metacharacters, and *regular expression* patterns have all the double-quote metacharacters plus some extra ones of their own.

metasymbol

Something we'd call a *metacharacter* except that it's a sequence of more than one character. Generally, the first character in the sequence must be a true metacharacter to get the other characters in the metasymbol to misbehave along with it.

method

A kind of action that an *object* can take if you tell it to. See *perlobj*.

minimalism

The belief that "small is beautiful." Paradoxically, if you say something in a small language, it turns out big, and if you say it in a big language, it turns out small. Go figure.

mode

In the context of the *stat* syscall, refers to the field holding the *permission bits* and the type of the *file*.

modifier

See *statement modifier*, *regular expression modifier*, and *lvalue modifier*, not necessarily in that order.

module

A *file* that defines a *package* of (almost) the same name, which can either *export* symbols or function as an *object* class. (A module's main *.pm* file may also load in other files in support of the module.) See the *use* built-in.

modulus

An integer divisor when you're interested in the remainder instead of the quotient.

monger

Short for Perl Monger, a purveyor of Perl.

mortal

A temporary value scheduled to die when the current statement finishes.

multidimensional array

An array with multiple subscripts for finding a single element. Perl implements these using *references*--see *perllol* and *perldsc*.

multiple inheritance

The features you got from your mother and father, mixed together unpredictably. (See also *inheritance*, and *single inheritance*.) In computer languages (including Perl), the notion that a given class may have multiple direct ancestors or *base classes*.

N

named pipe

A *pipe* with a name embedded in the *filesystem* so that it can be accessed by two unrelated

processes.

namespace

A domain of names. You needn't worry about whether the names in one such domain have been used in another. See *package*.

network address

The most important attribute of a socket, like your telephone's telephone number. Typically an IP address. See also *port*.

newline

A single character that represents the end of a line, with the ASCII value of 012 octal under Unix (but 015 on a Mac), and represented by \n in Perl strings. For Windows machines writing text files, and for certain physical devices like terminals, the single newline gets automatically translated by your C library into a line feed and a carriage return, but normally, no translation is done.

NFS

Network File System, which allows you to mount a remote filesystem as if it were local.

null character

A character with the ASCII value of zero. It's used by C to terminate strings, but Perl allows strings to contain a null.

null list

A *list value* with zero elements, represented in Perl by ( ).

null string

A *string* containing no characters, not to be confused with a string containing a *null character*, which has a positive length and is *true*.

numeric context

The situation in which an expression is expected by its surroundings (the code calling it) to return a number. See also *context* and *string context*.

NV

Short for Nevada, no part of which will ever be confused with civilization. NV also means an internal floating-point Numeric Value of the type a *scalar* can hold, not to be confused with an *IV*.

nybble

Half a *byte*, equivalent to one *hexadecimal* digit, and worth four *bits*.

O

object

An *instance* of a *class*. Something that "knows" what user-defined type (class) it is, and what it can do because of what class it is. Your program can request an object to do things, but the object gets to decide whether it wants to do them or not. Some objects are more accommodating than others.

octal

A number in base 8. Only the digits 0 through 7 are allowed. Octal constants in Perl start with 0, as in 013. See also the *oct* function.

offset

How many things you have to skip over when moving from the beginning of a string or array to

a specific position within it. Thus, the minimum offset is zero, not one, because you don't skip anything to get to the first item.

one-liner

An entire computer program crammed into one line of text.

open source software

Programs for which the source code is freely available and freely redistributable, with no commercial strings attached. For a more detailed definition, see *http://www.opensource.org/osd.html*.

operand

An *expression* that yields a *value* that an *operator* operates on. See also *precedence*.

operating system

A special program that runs on the bare machine and hides the gory details of managing *processes* and *devices*. Usually used in a looser sense to indicate a particular culture of programming. The loose sense can be used at varying levels of specificity. At one extreme, you might say that all versions of Unix and Unix-lookalikes are the same operating system (upsetting many people, especially lawyers and other advocates). At the other extreme, you could say this particular version of this particular vendor's operating system is different from any other version of this or any other vendor's operating system. Perl is much more portable across operating systems than many other languages. See also *architecture* and *platform*.

operator

A gizmo that transforms some number of input values to some number of output values, often built into a language with a special syntax or symbol. A given operator may have specific expectations about what *types* of data you give as its arguments (*operands*) and what type of data you want back from it.

operator overloading

A kind of *overloading* that you can do on built-in *operators* to make them work on *objects* as if the objects were ordinary scalar values, but with the actual semantics supplied by the object class. This is set up with the *overload pragma*.

options

See either *switches* or *regular expression modifier*.

ordinal

Another name for *code point*

overloading

Giving additional meanings to a symbol or construct. Actually, all languages do overloading to one extent or another, since people are good at figuring out things from *context*.

overriding

Hiding or invalidating some other definition of the same name. (Not to be confused with *overloading*, which adds definitions that must be disambiguated some other way.) To confuse the issue further, we use the word with two overloaded definitions: to describe how you can define your own *subroutine* to hide a built-in *function* of the same name (see *"Overriding Built-in Functions" in perlsub*) and to describe how you can define a replacement *method* in a *derived class* to hide a *base class*'s method of the same name (see *perlobj*).

owner

The one user (apart from the superuser) who has absolute control over a *file*. A file may also have a *group* of users who may exercise joint ownership if the real owner permits it. See

permission bits.

P

package

A *namespace* for global *variables*, *subroutines*, and the like, such that they can be kept separate from like-named *symbols* in other namespaces. In a sense, only the package is global, since the symbols in the package's symbol table are only accessible from code compiled outside the package by naming the package. But in another sense, all package symbols are also globals--they're just well-organized globals.

pad

Short for *scratchpad*.

parameter

See *argument*.

parent class

See *base class*.

parse tree

See *syntax tree*.

parsing

The subtle but sometimes brutal art of attempting to turn your possibly malformed program into a valid *syntax tree*.

patch

To fix by applying one, as it were. In the realm of hackerdom, a listing of the differences between two versions of a program as might be applied by the *patch*(1) program when you want to fix a bug or upgrade your old version.

PATH

The list of *directories* the system searches to find a program you want to *execute*. The list is stored as one of your *environment variables*, accessible in Perl as `$ENV{PATH}`.

pathname

A fully qualified filename such as */usr/bin/perl*. Sometimes confused with *PATH*.

pattern

A template used in *pattern matching*.

pattern matching

Taking a pattern, usually a *regular expression*, and trying the pattern various ways on a string to see whether there's any way to make it fit. Often used to pick interesting tidbits out of a file.

permission bits

Bits that the *owner* of a file sets or unsets to allow or disallow access to other people. These flag bits are part of the *mode* word returned by the *stat* built-in when you ask about a file. On Unix systems, you can check the *ls*(1) manpage for more information.

Pern

What you get when you do `Perl++` twice. Doing it only once will curl your hair. You have to increment it eight times to shampoo your hair. Lather, rinse, iterate.

pipe

A direct *connection* that carries the output of one *process* to the input of another without an

intermediate temporary file. Once the pipe is set up, the two processes in question can read and write as if they were talking to a normal file, with some caveats.

pipeline

A series of *processes* all in a row, linked by *pipes*, where each passes its output stream to the next.

platform

The entire hardware and software context in which a program runs. A program written in a platform-dependent language might break if you change any of: machine, operating system, libraries, compiler, or system configuration. The *perl* interpreter has to be compiled differently for each platform because it is implemented in C, but programs written in the Perl language are largely platform-independent.

pod

The markup used to embed documentation into your Perl code. See *perlpod*.

pointer

A *variable* in a language like C that contains the exact memory location of some other item. Perl handles pointers internally so you don't have to worry about them. Instead, you just use symbolic pointers in the form of *keys* and *variable* names, or *hard references*, which aren't pointers (but act like pointers and do in fact contain pointers).

polymorphism

The notion that you can tell an *object* to do something generic, and the object will interpret the command in different ways depending on its type. [<Gk many shapes]

port

The part of the address of a TCP or UDP socket that directs packets to the correct process after finding the right machine, something like the phone extension you give when you reach the company operator. Also, the result of converting code to run on a different platform than originally intended, or the verb denoting this conversion.

portable

Once upon a time, C code compilable under both BSD and SysV. In general, code that can be easily converted to run on another *platform*, where "easily" can be defined however you like, and usually is. Anything may be considered portable if you try hard enough. See *mobile home* or *London Bridge*.

porter

Someone who "carries" software from one *platform* to another. Porting programs written in platform-dependent languages such as C can be difficult work, but porting programs like Perl is very much worth the agony.

POSIX

The Portable Operating System Interface specification.

postfix

An *operator* that follows its *operand*, as in `$x++`.

pp

An internal shorthand for a "push-pop" code, that is, C code implementing Perl's stack machine.

pragma

A standard module whose practical hints and suggestions are received (and possibly ignored)

at compile time. Pragmas are named in all lowercase.

precedence

The rules of conduct that, in the absence of other guidance, determine what should happen first. For example, in the absence of parentheses, you always do multiplication before addition.

prefix

An *operator* that precedes its *operand*, as in ++$x.

preprocessing

What some helper *process* did to transform the incoming data into a form more suitable for the current process. Often done with an incoming *pipe*. See also *C preprocessor*.

procedure

A *subroutine*.

process

An instance of a running program. Under multitasking systems like Unix, two or more separate processes could be running the same program independently at the same time--in fact, the *fork* function is designed to bring about this happy state of affairs. Under other operating systems, processes are sometimes called "threads", "tasks", or "jobs", often with slight nuances in meaning.

program generator

A system that algorithmically writes code for you in a high-level language. See also *code generator*.

progressive matching

Pattern matching that picks up where it left off before.

property

See either *instance variable* or *character property*.

protocol

In networking, an agreed-upon way of sending messages back and forth so that neither correspondent will get too confused.

prototype

An optional part of a *subroutine* declaration telling the Perl compiler how many and what flavor of arguments may be passed as *actual arguments*, so that you can write subroutine calls that parse much like built-in functions. (Or don't parse, as the case may be.)

pseudofunction

A construct that sometimes looks like a function but really isn't. Usually reserved for *lvalue* modifiers like *my*, for *context* modifiers like *scalar*, and for the pick-your-own-quotes constructs, `q//`, `qq//`, `qx//`, `qw//`, `qr//`, `m//`, `s///`, `y///`, and `tr///`.

pseudohash

A reference to an array whose initial element happens to hold a reference to a hash. You can treat a pseudohash reference as either an array reference or a hash reference.

pseudoliteral

An *operator* that looks something like a *literal*, such as the output-grabbing operator, `` `command` ``.

public domain

Something not owned by anybody. Perl is copyrighted and is thus *not* in the public domain--it's just *freely available* and *freely redistributable*.

pumpkin

A notional "baton" handed around the Perl community indicating who is the lead integrator in some arena of development.

pumpking

A *pumpkin* holder, the person in charge of pumping the pump, or at least priming it. Must be willing to play the part of the Great Pumpkin now and then.

PV

A "pointer value", which is Perl Internals Talk for a `char*`.

Q

qualified

Possessing a complete name. The symbol `$Ent::moot` is qualified; `$moot` is unqualified. A fully qualified filename is specified from the top-level directory.

quantifier

A component of a *regular expression* specifying how many times the foregoing *atom* may occur.

R

readable

With respect to files, one that has the proper permission bit set to let you access the file. With respect to computer programs, one that's written well enough that someone has a chance of figuring out what it's trying to do.

reaping

The last rites performed by a parent *process* on behalf of a deceased child process so that it doesn't remain a *zombie*. See the *wait* and *waitpid* function calls.

record

A set of related data values in a *file* or *stream*, often associated with a unique *key* field. In Unix, often commensurate with a *line*, or a blank-line-terminated set of lines (a "paragraph"). Each line of the */etc/passwd* file is a record, keyed on login name, containing information about that user.

recursion

The art of defining something (at least partly) in terms of itself, which is a naughty no-no in dictionaries but often works out okay in computer programs if you're careful not to recurse forever, which is like an infinite loop with more spectacular failure modes.

reference

Where you look to find a pointer to information somewhere else. (See *indirection*.) References come in two flavors, *symbolic references* and *hard references*.

referent

Whatever a reference refers to, which may or may not have a name. Common types of referents include scalars, arrays, hashes, and subroutines.

regex

See *regular expression*.

regular expression

A single entity with various interpretations, like an elephant. To a computer scientist, it's a grammar for a little language in which some strings are legal and others aren't. To normal people, it's a pattern you can use to find what you're looking for when it varies from case to case. Perl's regular expressions are far from regular in the theoretical sense, but in regular use they work quite well. Here's a regular expression: `/Oh s.*t./`. This will match strings like "`Oh say can you see by the dawn's early light`" and "`Oh sit!`". See *perlre*.

regular expression modifier

An option on a pattern or substitution, such as `/i` to render the pattern case insensitive. See also *cloister*.

regular file

A *file* that's not a *directory*, a *device*, a named *pipe* or *socket*, or a *symbolic link*. Perl uses the `-f` file test operator to identify regular files. Sometimes called a "plain" file.

relational operator

An *operator* that says whether a particular ordering relationship is *true* about a pair of *operands*. Perl has both numeric and string relational operators. See *collating sequence*.

reserved words

A word with a specific, built-in meaning to a *compiler*, such as `if` or *delete*. In many languages (not Perl), it's illegal to use reserved words to name anything else. (Which is why they're reserved, after all.) In Perl, you just can't use them to name *labels* or *filehandles*. Also called "keywords".

return value

The *value* produced by a *subroutine* or *expression* when evaluated. In Perl, a return value may be either a *list* or a *scalar*.

RFC

Request For Comment, which despite the timid connotations is the name of a series of important standards documents.

right shift

A *bit shift* that divides a number by some power of 2.

root

The superuser (UID == 0). Also, the top-level directory of the filesystem.

RTFM

What you are told when someone thinks you should Read The Fine Manual.

run phase

Any time after Perl starts running your main program. See also *compile phase*. Run phase is mostly spent in *run time* but may also be spent in *compile time* when *require*, *do* `FILE`, or *eval* `STRING` operators are executed or when a substitution uses the `/ee` modifier.

run time

The time when Perl is actually doing what your code says to do, as opposed to the earlier period of time when it was trying to figure out whether what you said made any sense whatsoever, which is *compile time*.

run-time pattern

A pattern that contains one or more variables to be interpolated before parsing the pattern as a *regular expression*, and that therefore cannot be analyzed at compile time, but must be

re-analyzed each time the pattern match operator is evaluated. Run-time patterns are useful but expensive.

RV

A recreational vehicle, not to be confused with vehicular recreation. RV also means an internal Reference Value of the type a *scalar* can hold. See also *IV* and *NV* if you're not confused yet.

rvalue

A *value* that you might find on the right side of an *assignment*. See also *lvalue*.

S

scalar

A simple, singular value; a number, *string*, or *reference*.

scalar context

The situation in which an *expression* is expected by its surroundings (the code calling it) to return a single *value* rather than a *list* of values. See also *context* and *list context*. A scalar context sometimes imposes additional constraints on the return value--see *string context* and *numeric context*. Sometimes we talk about a *Boolean context* inside conditionals, but this imposes no additional constraints, since any scalar value, whether numeric or *string*, is already true or false.

scalar literal

A number or quoted *string*--an actual *value* in the text of your program, as opposed to a *variable*.

scalar value

A value that happens to be a *scalar* as opposed to a *list*.

scalar variable

A *variable* prefixed with $ that holds a single value.

scope

How far away you can see a variable from, looking through one. Perl has two visibility mechanisms: it does *dynamic scoping* of *local variables*, meaning that the rest of the *block*, and any *subroutines* that are called by the rest of the block, can see the variables that are local to the block. Perl does *lexical scoping* of *my* variables, meaning that the rest of the block can see the variable, but other subroutines called by the block *cannot* see the variable.

scratchpad

The area in which a particular invocation of a particular file or subroutine keeps some of its temporary values, including any lexically scoped variables.

script

A text *file* that is a program intended to be *executed* directly rather than *compiled* to another form of file before execution. Also, in the context of *Unicode*, a writing system for a particular language or group of languages, such as Greek, Bengali, or Klingon.

script kiddie

A *cracker* who is not a *hacker*, but knows just enough to run canned scripts. A cargo-cult programmer.

sed

A venerable Stream EDitor from which Perl derives some of its ideas.

semaphore

A fancy kind of interlock that prevents multiple *threads* or *processes* from using up the same resources simultaneously.

separator

A *character* or *string* that keeps two surrounding strings from being confused with each other. The *split* function works on separators. Not to be confused with *delimiters* or *terminators*. The "or" in the previous sentence separated the two alternatives.

serialization

Putting a fancy *data structure* into linear order so that it can be stored as a *string* in a disk file or database or sent through a *pipe*. Also called marshalling.

server

In networking, a *process* that either advertises a *service* or just hangs around at a known location and waits for *clients* who need service to get in touch with it.

service

Something you do for someone else to make them happy, like giving them the time of day (or of their life). On some machines, well-known services are listed by the *getservent* function.

setgid

Same as *setuid*, only having to do with giving away *group* privileges.

setuid

Said of a program that runs with the privileges of its *owner* rather than (as is usually the case) the privileges of whoever is running it. Also describes the bit in the mode word (*permission bits*) that controls the feature. This bit must be explicitly set by the owner to enable this feature, and the program must be carefully written not to give away more privileges than it ought to.

shared memory

A piece of *memory* accessible by two different *processes* who otherwise would not see each other's memory.

shebang

Irish for the whole McGillicuddy. In Perl culture, a portmanteau of "sharp" and "bang", meaning the `#!` sequence that tells the system where to find the interpreter.

shell

A *command*-line *interpreter*. The program that interactively gives you a prompt, accepts one or more *lines* of input, and executes the programs you mentioned, feeding each of them their proper *arguments* and input data. Shells can also execute scripts containing such commands. Under Unix, typical shells include the Bourne shell (*/bin/sh*), the C shell (*/bin/csh*), and the Korn shell (*/bin/ksh*). Perl is not strictly a shell because it's not interactive (although Perl programs can be interactive).

side effects

Something extra that happens when you evaluate an *expression*. Nowadays it can refer to almost anything. For example, evaluating a simple assignment statement typically has the "side effect" of assigning a value to a variable. (And you thought assigning the value was your primary intent in the first place!) Likewise, assigning a value to the special variable `$|` (`$AUTOFLUSH`) has the side effect of forcing a flush after every *write* or *print* on the currently selected filehandle.

signal

A bolt out of the blue; that is, an event triggered by the *operating system*, probably when you're least expecting it.

signal handler

A *subroutine* that, instead of being content to be called in the normal fashion, sits around waiting for a bolt out of the blue before it will deign to *execute*. Under Perl, bolts out of the blue are called signals, and you send them with the *kill* built-in. See *"%SIG" in perlvar* and *"Signals" in perlipc*.

single inheritance

The features you got from your mother, if she told you that you don't have a father. (See also *inheritance* and *multiple inheritance*.) In computer languages, the notion that *classes* reproduce asexually so that a given class can only have one direct ancestor or *base class*. Perl supplies no such restriction, though you may certainly program Perl that way if you like.

slice

A selection of any number of *elements* from a *list*, *array*, or *hash*.

slurp

To read an entire *file* into a *string* in one operation.

socket

An endpoint for network communication among multiple *processes* that works much like a telephone or a post office box. The most important thing about a socket is its *network address* (like a phone number). Different kinds of sockets have different kinds of addresses--some look like filenames, and some don't.

soft reference

See *symbolic reference*.

source filter

A special kind of *module* that does *preprocessing* on your script just before it gets to the *tokener*.

stack

A device you can put things on the top of, and later take them back off in the opposite order in which you put them on. See *LIFO*.

standard

Included in the official Perl distribution, as in a standard module, a standard tool, or a standard Perl *manpage*.

standard error

The default output *stream* for nasty remarks that don't belong in *standard output*. Represented within a Perl program by the *filehandle STDERR*. You can use this stream explicitly, but the *die* and *warn* built-ins write to your standard error stream automatically.

standard I/O

A standard C library for doing *buffered* input and output to the *operating system*. (The "standard" of standard I/O is only marginally related to the "standard" of standard input and output.) In general, Perl relies on whatever implementation of standard I/O a given operating system supplies, so the buffering characteristics of a Perl program on one machine may not exactly match those on another machine. Normally this only influences efficiency, not semantics. If your standard I/O package is doing block buffering and you want it to *flush* the buffer more often, just set the $| variable to a true value.

standard input

The default input *stream* for your program, which if possible shouldn't care where its data is coming from. Represented within a Perl program by the *filehandle STDIN*.

standard output

The default output *stream* for your program, which if possible shouldn't care where its data is going. Represented within a Perl program by the *filehandle STDOUT*.

stat structure

A special internal spot in which Perl keeps the information about the last *file* on which you requested information.

statement

A *command* to the computer about what to do next, like a step in a recipe: "Add marmalade to batter and mix until mixed." A statement is distinguished from a *declaration*, which doesn't tell the computer to do anything, but just to learn something.

statement modifier

A *conditional* or *loop* that you put after the *statement* instead of before, if you know what we mean.

static

Varying slowly compared to something else. (Unfortunately, everything is relatively stable compared to something else, except for certain elementary particles, and we're not so sure about them.) In computers, where things are supposed to vary rapidly, "static" has a derogatory connotation, indicating a slightly dysfunctional *variable*, *subroutine*, or *method*. In Perl culture, the word is politely avoided.

static method

No such thing. See *class method*.

static scoping

No such thing. See *lexical scoping*.

static variable

No such thing. Just use a *lexical variable* in a scope larger than your *subroutine*.

status

The *value* returned to the parent *process* when one of its child processes dies. This value is placed in the special variable $?. Its upper eight *bits* are the exit status of the defunct process, and its lower eight bits identify the signal (if any) that the process died from. On Unix systems, this status value is the same as the status word returned by *wait*(2). See *"system" in perlfunc*.

STDERR

See *standard error*.

STDIN

See *standard input*.

STDIO

See *standard I/O*.

STDOUT

See *standard output*.

stream

A flow of data into or out of a process as a steady sequence of bytes or characters, without the appearance of being broken up into packets. This is a kind of *interface*--the underlying *implementation* may well break your data up into separate packets for delivery, but this is hidden from you.

string

A sequence of characters such as "He said !@#*&%@#*?!". A string does not have to be entirely printable.

string context

The situation in which an expression is expected by its surroundings (the code calling it) to return a *string*. See also *context* and *numeric context*.

stringification

The process of producing a *string* representation of an abstract object.

struct

C keyword introducing a structure definition or name.

structure

See *data structure*.

subclass

See *derived class*.

subpattern

A component of a *regular expression* pattern.

subroutine

A named or otherwise accessible piece of program that can be invoked from elsewhere in the program in order to accomplish some sub-goal of the program. A subroutine is often parameterized to accomplish different but related things depending on its input *arguments*. If the subroutine returns a meaningful *value*, it is also called a *function*.

subscript

A *value* that indicates the position of a particular *array element* in an array.

substitution

Changing parts of a string via the `s///` operator. (We avoid use of this term to mean *variable interpolation*.)

substring

A portion of a *string*, starting at a certain *character* position (*offset*) and proceeding for a certain number of characters.

superclass

See *base class*.

superuser

The person whom the *operating system* will let do almost anything. Typically your system administrator or someone pretending to be your system administrator. On Unix systems, the *root* user. On Windows systems, usually the Administrator user.

SV

Short for "scalar value". But within the Perl interpreter every *referent* is treated as a member of a class derived from SV, in an object-oriented sort of way. Every *value* inside Perl is passed around as a C language `SV*` pointer. The SV *struct* knows its own "referent type", and the code is smart enough (we hope) not to try to call a *hash* function on a *subroutine*.

switch

An option you give on a command line to influence the way your program works, usually

introduced with a minus sign. The word is also used as a nickname for a *switch statement*.

switch cluster

The combination of multiple command-line switches (e.g., **-a -b -c**) into one switch (e.g., **-abc**). Any switch with an additional *argument* must be the last switch in a cluster.

switch statement

A program technique that lets you evaluate an *expression* and then, based on the value of the expression, do a multiway branch to the appropriate piece of code for that value. Also called a "case structure", named after the similar Pascal construct. See *"Switch statements" in perlsyn*.

symbol

Generally, any *token* or *metasymbol*. Often used more specifically to mean the sort of name you might find in a *symbol table*.

symbol table

Where a *compiler* remembers symbols. A program like Perl must somehow remember all the names of all the *variables*, *filehandles*, and *subroutines* you've used. It does this by placing the names in a symbol table, which is implemented in Perl using a *hash table*. There is a separate symbol table for each *package* to give each package its own *namespace*.

symbolic debugger

A program that lets you step through the *execution* of your program, stopping or printing things out here and there to see whether anything has gone wrong, and if so, what. The "symbolic" part just means that you can talk to the debugger using the same symbols with which your program is written.

symbolic link

An alternate filename that points to the real *filename*, which in turn points to the real *file*. Whenever the *operating system* is trying to parse a *pathname* containing a symbolic link, it merely substitutes the new name and continues parsing.

symbolic reference

A variable whose value is the name of another variable or subroutine. By *dereferencing* the first variable, you can get at the second one. Symbolic references are illegal under *use strict 'refs'*.

synchronous

Programming in which the orderly sequence of events can be determined; that is, when things happen one after the other, not at the same time.

syntactic sugar

An alternative way of writing something more easily; a shortcut.

syntax

From Greek, "with-arrangement". How things (particularly symbols) are put together with each other.

syntax tree

An internal representation of your program wherein lower-level *constructs* dangle off the higher-level constructs enclosing them.

syscall

A *function* call directly to the *operating system*. Many of the important subroutines and functions you use aren't direct system calls, but are built up in one or more layers above the system call level. In general, Perl programmers don't need to worry about the distinction.

However, if you do happen to know which Perl functions are really syscalls, you can predict which of these will set the `$!` (`$ERRNO`) variable on failure. Unfortunately, beginning programmers often confusingly employ the term "system call" to mean what happens when you call the Perl *system* function, which actually involves many syscalls. To avoid any confusion, we nearly always use say "syscall" for something you could call indirectly via Perl's *syscall* function, and never for something you would call with Perl's *system* function.

T

tainted

Said of data derived from the grubby hands of a user and thus unsafe for a secure program to rely on. Perl does taint checks if you run a *setuid* (or *setgid*) program, or if you use the **-T** switch.

TCP

Short for Transmission Control Protocol. A protocol wrapped around the Internet Protocol to make an unreliable packet transmission mechanism appear to the application program to be a reliable *stream* of bytes. (Usually.)

term

Short for a "terminal", that is, a leaf node of a *syntax tree*. A thing that functions grammatically as an *operand* for the operators in an expression.

terminator

A *character* or *string* that marks the end of another string. The `$/` variable contains the string that terminates a *readline* operation, which *chomp* deletes from the end. Not to be confused with *delimiters* or *separators*. The period at the end of this sentence is a terminator.

ternary

An *operator* taking three *operands*. Sometimes pronounced *trinary*.

text

A *string* or *file* containing primarily printable characters.

thread

Like a forked process, but without *fork*'s inherent memory protection. A thread is lighter weight than a full process, in that a process could have multiple threads running around in it, all fighting over the same process's memory space unless steps are taken to protect threads from each other. See *threads*.

tie

The bond between a magical variable and its implementation class. See *"tie" in perlfunc* and *perltie*.

TMTOWTDI

There's More Than One Way To Do It, the Perl Motto. The notion that there can be more than one valid path to solving a programming problem in context. (This doesn't mean that more ways are always better or that all possible paths are equally desirable--just that there need not be One True Way.) Pronounced TimToady.

token

A morpheme in a programming language, the smallest unit of text with semantic significance.

tokener

A module that breaks a program text into a sequence of *tokens* for later analysis by a parser.

tokenizing

Splitting up a program text into *tokens*. Also known as "lexing", in which case you get "lexemes" instead of tokens.

toolbox approach

The notion that, with a complete set of simple tools that work well together, you can build almost anything you want. Which is fine if you're assembling a tricycle, but if you're building a defranishizing comboflux regurgalator, you really want your own machine shop in which to build special tools. Perl is sort of a machine shop.

transliterate

To turn one string representation into another by mapping each character of the source string to its corresponding character in the result string. See *"tr/SEARCHLIST/REPLACEMENTLIST/cdsr" in perlop*.

trigger

An event that causes a *handler* to be run.

trinary

Not a stellar system with three stars, but an *operator* taking three *operands*. Sometimes pronounced *ternary*.

troff

A venerable typesetting language from which Perl derives the name of its $% variable and which is secretly used in the production of Camel books.

true

Any scalar value that doesn't evaluate to 0 or `""`.

truncating

Emptying a file of existing contents, either automatically when opening a file for writing or explicitly via the *truncate* function.

type

See *data type* and *class*.

type casting

Converting data from one type to another. C permits this. Perl does not need it. Nor want it.

typed lexical

A *lexical variable* that is declared with a *class* type: `my Pony $bill`.

typedef

A type definition in the C language.

typeglob

Use of a single identifier, prefixed with `*`. For example, `*name` stands for any or all of `$name`, `@name`, `%name`, `&name`, or just `name`. How you use it determines whether it is interpreted as all or only one of them. See *"Typeglobs and Filehandles" in perldata*.

typemap

A description of how C types may be transformed to and from Perl types within an *extension* module written in *XS*.

U

UDP

User Datagram Protocol, the typical way to send *datagrams* over the Internet.

UID

A user ID. Often used in the context of *file* or *process* ownership.

umask

A mask of those *permission bits* that should be forced off when creating files or directories, in order to establish a policy of whom you'll ordinarily deny access to. See the *umask* function.

unary operator

An operator with only one *operand*, like ! or *chdir*. Unary operators are usually prefix operators; that is, they precede their operand. The ++ and -- operators can be either prefix or postfix. (Their position *does* change their meanings.)

Unicode

A character set comprising all the major character sets of the world, more or less. See *perlunicode* and *http://www.unicode.org*.

Unix

A very large and constantly evolving language with several alternative and largely incompatible syntaxes, in which anyone can define anything any way they choose, and usually do. Speakers of this language think it's easy to learn because it's so easily twisted to one's own ends, but dialectical differences make tribal intercommunication nearly impossible, and travelers are often reduced to a pidgin-like subset of the language. To be universally understood, a Unix shell programmer must spend years of study in the art. Many have abandoned this discipline and now communicate via an Esperanto-like language called Perl.

In ancient times, Unix was also used to refer to some code that a couple of people at Bell Labs wrote to make use of a PDP-7 computer that wasn't doing much of anything else at the time.

V

value

An actual piece of data, in contrast to all the variables, references, keys, indexes, operators, and whatnot that you need to access the value.

variable

A named storage location that can hold any of various kinds of *value*, as your program sees fit.

variable interpolation

The *interpolation* of a scalar or array variable into a string.

variadic

Said of a *function* that happily receives an indeterminate number of *actual arguments*.

vector

Mathematical jargon for a list of *scalar values*.

virtual

Providing the appearance of something without the reality, as in: virtual memory is not real memory. (See also *memory*.) The opposite of "virtual" is "transparent", which means providing the reality of something without the appearance, as in: Perl handles the variable-length UTF-8 character encoding transparently.

void context

A form of *scalar context* in which an *expression* is not expected to return any *value* at all and is evaluated for its *side effects* alone.

v-string

A "version" or "vector" *string* specified with a v followed by a series of decimal integers in dot notation, for instance, `v1.20.300.4000`. Each number turns into a *character* with the specified ordinal value. (The v is optional when there are at least three integers.)

W

warning

A message printed to the *STDERR* stream to the effect that something might be wrong but isn't worth blowing up over. See *"warn" in perlfunc* and the *warnings* pragma.

watch expression

An expression which, when its value changes, causes a breakpoint in the Perl debugger.

whitespace

A *character* that moves your cursor but doesn't otherwise put anything on your screen. Typically refers to any of: space, tab, line feed, carriage return, or form feed.

word

In normal "computerese", the piece of data of the size most efficiently handled by your computer, typically 32 bits or so, give or take a few powers of 2. In Perl culture, it more often refers to an alphanumeric *identifier* (including underscores), or to a string of nonwhitespace *characters* bounded by whitespace or string boundaries.

working directory

Your current *directory*, from which relative pathnames are interpreted by the *operating system*. The operating system knows your current directory because you told it with a *chdir* or because you started out in the place where your parent *process* was when you were born.

wrapper

A program or subroutine that runs some other program or subroutine for you, modifying some of its input or output to better suit your purposes.

WYSIWYG

What You See Is What You Get. Usually used when something that appears on the screen matches how it will eventually look, like Perl's *format* declarations. Also used to mean the opposite of magic because everything works exactly as it appears, as in the three-argument form of *open*.

X

XS

An extraordinarily exported, expeditiously excellent, expressly eXternal Subroutine, executed in existing C or C++ or in an exciting new extension language called (exasperatingly) XS. Examine *perlxs* for the exact explanation or *perlxstut* for an exemplary unexacting one.

XSUB

An external *subroutine* defined in *XS*.

Y

yacc

Yet Another Compiler Compiler. A parser generator without which Perl probably would not have existed. See the file *perly.y* in the Perl source distribution.

Z

zero width

A subpattern *assertion* matching the *null string* between *characters*.

zombie

A process that has died (exited) but whose parent has not yet received proper notification of its demise by virtue of having called *wait* or *waitpid*. If you *fork*, you must clean up after your child processes when they exit, or else the process table will fill up and your system administrator will Not Be Happy with you.

AUTHOR AND COPYRIGHT