# NAME

perldiag - various Perl diagnostics

# DESCRIPTION

These messages are classified as follows (listed in increasing order of desperation):

```
(W) A warning (optional).
(D) A deprecation (optional).
(S) A severe warning (enabled by default).
(F) A fatal error (trappable).
(P) An internal error you should never see (trappable).
(X) A very fatal error (nontrappable).
(A) An alien error message (not generated by Perl).
```

The majority of messages from the first three classifications above (W, D & S) can be controlled using the `warnings` pragma.

If a message can be controlled by the `warnings` pragma, its warning category is included with the classification letter in the description below.

Optional warnings are enabled by using the `warnings` pragma or the **-w** and **-W** switches. Warnings may be captured by setting `$SIG{__WARN__}` to a reference to a routine that will be called on each warning instead of printing it. See *perlvar*.

Severe warnings are always enabled, unless they are explicitly disabled with the `warnings` pragma or the **-X** switch.

Trappable errors may be trapped using the eval operator. See *"eval" in perlfunc*. In almost all cases, warnings may be selectively disabled or promoted to fatal errors using the `warnings` pragma. See *warnings*.

The messages are in alphabetical order, without regard to upper or lower-case. Some of these messages are generic. Spots that vary are denoted with a %s or other printf-style escape. These escapes are ignored by the alphabetical order, as are all characters other than letters. To look up your message, just ignore anything that is not a letter.

accept() on closed socket %s

> (W closed) You tried to do an accept on a closed socket. Did you forget to check the return value of your socket() call? See *"accept" in perlfunc*.

Allocation too large: %lx

> (X) You can't allocate more than 64K on an MS-DOS machine.

'%c' allowed only after types %s

> (F) The modifiers '!', '<' and '>' are allowed in pack() or unpack() only after certain types. See *"pack" in perlfunc*.

Ambiguous call resolved as CORE::%s(), qualify as such or use &

> (W ambiguous) A subroutine you have declared has the same name as a Perl keyword, and you have used the name without qualification for calling one or the other. Perl decided to call the builtin because the subroutine is not imported.
>
> To force interpretation as a subroutine call, either put an ampersand before the subroutine name, or qualify the name with its package. Alternatively, you can import the subroutine (or pretend that it's imported with the `use subs` pragma).
>
> To silently interpret it as the Perl operator, use the `CORE::` prefix on the operator (e.g. `CORE::log($x)`) or declare the subroutine to be an object method (see *"Subroutine Attributes" in perlsub* or *attributes*).

Ambiguous range in transliteration operator

> (F) You wrote something like `tr/a-z-0//` which doesn't mean anything at all. To include a `-` character in a transliteration, put it either first or last. (In the past, `tr/a-z-0//` was synonymous with `tr/a-y//`, which was probably not what you would have expected.)

Ambiguous use of %s resolved as %s

> (W ambiguous)(S) You said something that may not be interpreted the way you thought. Normally it's pretty easy to disambiguate it by supplying a missing quote, operator, parenthesis pair or declaration.

'|' and '<' may not both be specified on command line

> (F) An error peculiar to VMS. Perl does its own command line redirection, and found that STDIN was a pipe, and that you also tried to redirect STDIN using '<'. Only one STDIN stream to a customer, please.

'|' and '>' may not both be specified on command line

> (F) An error peculiar to VMS. Perl does its own command line redirection, and thinks you tried to redirect stdout both to a file and into a pipe to another command. You need to choose one or the other, though nothing's stopping you from piping into a program or Perl script which 'splits' output into two streams, such as
>
> ```
> open(OUT,">$ARGV[0]") or die "Can't write to $ARGV[0]: $!";
> while (<STDIN>) {
>     print;
>     print OUT;
> }
> close OUT;
> ```

Applying %s to %s will act on scalar(%s)

> (W misc) The pattern match (`//`), substitution (`s///`), and transliteration (`tr///`) operators work on scalar values. If you apply one of them to an array or a hash, it will convert the array or hash to a scalar value -- the length of an array, or the population info of a hash -- and then work on that scalar value. This is probably not what you meant to do. See *"grep" in perlfunc* and *"map" in perlfunc* for alternatives.

Args must match #! line

> (F) The setuid emulator requires that the arguments Perl was invoked with match the arguments specified on the #! line. Since some systems impose a one-argument limit on the #! line, try combining switches; for example, turn `-w -U` into `-wU`.

Arg too short for msgsnd

> (F) msgsnd() requires a string at least as long as sizeof(long).

%s argument is not a HASH or ARRAY element or a subroutine

> (F) The argument to exists() must be a hash or array element or a subroutine with an ampersand, such as:
>
> ```
> $foo{$bar}
> $ref->{"susie"}[12]
> &do_something
> ```

%s argument is not a HASH or ARRAY element or slice

> (F) The argument to delete() must be either a hash or array element, such as:
>
> ```
> $foo{$bar}
> $ref->{"susie"}[12]
> ```

or a hash or array slice, such as:

```
@foo[$bar, $baz, $xyzzy]
@{$ref->[12]}{"susie", "queue"}
```

%s argument is not a subroutine name

(F) The argument to exists() for `exists &sub` must be a subroutine name, and not a subroutine call. `exists &sub()` will generate this error.

Argument "%s" isn't numeric%s

(W numeric) The indicated string was fed as an argument to an operator that expected a numeric value instead. If you're fortunate the message will identify which operator was so unfortunate.

Argument list not closed for PerlIO layer "%s"

(W layer) When pushing a layer with arguments onto the Perl I/O system you forgot the ) that closes the argument list. (Layers take care of transforming data between external and internal representations.) Perl stopped parsing the layer list at this point and did not attempt to push this layer. If your program didn't explicitly request the failing operation, it may be the result of the value of the environment variable PERLIO.

Array @%s missing the @ in argument %d of %s()

(D deprecated) Really old Perl let you omit the @ on array names in some spots. This is now heavily deprecated.

assertion botched: %s

(P) The malloc package that comes with Perl had an internal failure.

Assertion failed: file "%s"

(P) A general assertion failed. The file in question must be examined.

Assignment to both a list and a scalar

(F) If you assign to a conditional operator, the 2nd and 3rd arguments must either both be scalars or both be lists. Otherwise Perl won't know which context to supply to the right side.

A thread exited while %d threads were running

(W threads)(S) When using threaded Perl, a thread (not necessarily the main thread) exited while there were still other threads running. Usually it's a good idea to first collect the return values of the created threads by joining them, and only then exit from the main thread. See *threads*.

Attempt to access disallowed key '%s' in a restricted hash

(F) The failing code has attempted to get or set a key which is not in the current set of allowed keys of a restricted hash.

Attempt to bless into a reference

(F) The CLASSNAME argument to the bless() operator is expected to be the name of the package to bless the resulting object into. You've supplied instead a reference to something: perhaps you wrote

```
bless $self, $proto;
```

when you intended

```
bless $self, ref($proto) || $proto;
```

If you actually want to bless into the stringified version of the reference supplied, you need to

stringify it yourself, for example by:

```
bless $self, "$proto";
```

Attempt to delete disallowed key '%s' from a restricted hash

(F) The failing code attempted to delete from a restricted hash a key which is not in its key set.

Attempt to delete readonly key '%s' from a restricted hash

(F) The failing code attempted to delete a key whose value has been declared readonly from a restricted hash.

Attempt to free non-arena SV: 0x%lx

(P internal) All SV objects are supposed to be allocated from arenas that will be garbage collected on exit. An SV was discovered to be outside any of those arenas.

Attempt to free nonexistent shared string

(P internal) Perl maintains a reference counted internal table of strings to optimize the storage and access of hash keys and other strings. This indicates someone tried to decrement the reference count of a string that can no longer be found in the table.

Attempt to free temp prematurely

(W debugging) Mortalized values are supposed to be freed by the free_tmps() routine. This indicates that something else is freeing the SV before the free_tmps() routine gets a chance, which means that the free_tmps() routine will be freeing an unreferenced scalar when it does try to free it.

Attempt to free unreferenced glob pointers

(P internal) The reference counts got screwed up on symbol aliases.

Attempt to free unreferenced scalar

(W internal) Perl went to decrement the reference count of a scalar to see if it would go to 0, and discovered that it had already gone to 0 earlier, and should have been freed, and in fact, probably was freed. This could indicate that SvREFCNT_dec() was called too many times, or that SvREFCNT_inc() was called too few times, or that the SV was mortalized when it shouldn't have been, or that memory has been corrupted.

Attempt to join self

(F) You tried to join a thread from within itself, which is an impossible task. You may be joining the wrong thread, or you may need to move the join() to some other thread.

Attempt to pack pointer to temporary value

(W pack) You tried to pass a temporary value (like the result of a function, or a computed expression) to the "p" pack() template. This means the result contains a pointer to a location that could become invalid anytime, even before the end of the current statement. Use literals or global values as arguments to the "p" pack() template to avoid this warning.

Attempt to reload %s aborted.

(F) You tried to load a file with `use` or `require` that failed to compile once already. Perl will not try to compile this file again unless you delete its entry from %INC. See *"require" in perlfunc* and *"%INC" in perlvar*.

Attempt to set length of freed array

(W) You tried to set the length of an array which has been freed. You can do this by storing a reference to the scalar representing the last index of an array and later assigning through that reference. For example

```
$r = do {my @a; \$#a};
```

```
        $$r = 503
```

Attempt to use reference as lvalue in substr

> (W substr) You supplied a reference as the first argument to substr() used as an lvalue, which
> is pretty strange. Perhaps you forgot to dereference it first. See *"substr" in perlfunc*.

Bad arg length for %s, is %d, should be %s

> (F) You passed a buffer of the wrong size to one of msgctl(), semctl() or shmctl(). In C
> parlance, the correct sizes are, respectively, sizeof(struct msqid_ds *), sizeof(struct semid_ds
> *), and sizeof(struct shmid_ds *).

Bad evalled substitution pattern

> (F) You've used the `/e` switch to evaluate the replacement for a substitution, but perl found a
> syntax error in the code to evaluate, most likely an unexpected right brace '}'.

Bad filehandle: %s

> (F) A symbol was passed to something wanting a filehandle, but the symbol has no filehandle
> associated with it. Perhaps you didn't do an open(), or did it in another package.

Bad free() ignored

> (S malloc) An internal routine called free() on something that had never been malloc()ed in the
> first place. Mandatory, but can be disabled by setting environment variable `PERL_BADFREE` to
> 0.

> This message can be seen quite often with DB_File on systems with "hard" dynamic linking,
> like `AIX` and `OS/2`. It is a bug of `Berkeley DB` which is left unnoticed if `DB` uses *forgiving*
> system malloc().

Bad hash

> (P) One of the internal hash routines was passed a null HV pointer.

Badly placed ()'s

> (A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or
> manually feed your script into Perl yourself.

Bad name after %s::

> (F) You started to name a symbol by using a package prefix, and then didn't finish the symbol.
> In particular, you can't interpolate outside of quotes, so
>
> ```
>         $var = 'myvar';
>         $sym = mypack::$var;
> ```
>
> is not the same as
>
> ```
>         $var = 'myvar';
>         $sym = "mypack::$var";
> ```

Bad realloc() ignored

> (S malloc) An internal routine called realloc() on something that had never been malloc()ed in
> the first place. Mandatory, but can be disabled by setting environment variable
> `PERL_BADFREE` to 1.

Bad symbol for array

> (P) An internal request asked to add an array entry to something that wasn't a symbol table
> entry.

Bad symbol for dirhandle

(P) An internal request asked to add a dirhandle entry to something that wasn't a symbol table entry.

Bad symbol for filehandle

(P) An internal request asked to add a filehandle entry to something that wasn't a symbol table entry.

Bad symbol for hash

(P) An internal request asked to add a hash entry to something that wasn't a symbol table entry.

Bareword found in conditional

(W bareword) The compiler found a bareword where it expected a conditional, which often indicates that an || or && was parsed as part of the last argument of the previous construct, for example:

```
open FOO || die;
```

It may also indicate a misspelled constant that has been interpreted as a bareword:

```
use constant TYPO => 1;
if (TYOP) { print "foo" }
```

The strict pragma is useful in avoiding such errors.

Bareword "%s" not allowed while "strict subs" in use

(F) With "strict subs" in use, a bareword is only allowed as a subroutine identifier, in curly brackets or to the left of the "=>" symbol. Perhaps you need to predeclare a subroutine?

Bareword "%s" refers to nonexistent package

(W bareword) You used a qualified bareword of the form Foo::, but the compiler saw no other uses of that namespace before that point. Perhaps you need to predeclare a package?

BEGIN failed--compilation aborted

(F) An untrapped exception was raised while executing a BEGIN subroutine. Compilation stops immediately and the interpreter is exited.

BEGIN not safe after errors--compilation aborted

(F) Perl found a BEGIN {} subroutine (or a use directive, which implies a BEGIN {}) after one or more compilation errors had already occurred. Since the intended environment for the BEGIN {} could not be guaranteed (due to the errors), and since subsequent code likely depends on its correct operation, Perl just gave up.

\1 better written as $1

(W syntax) Outside of patterns, backreferences live on as variables. The use of backslashes is grandfathered on the right-hand side of a substitution, but stylistically it's better to use the variable form because other Perl programmers will expect it, and it works better if there are more than 9 backreferences.

Binary number > 0b11111111111111111111111111111111 non-portable

(W portable) The binary number you specified is larger than 2**32-1 (4294967295) and therefore non-portable between systems. See *perlport* for more on portability concerns.

bind() on closed socket %s

(W closed) You tried to do a bind on a closed socket. Did you forget to check the return value of your socket() call? See *"bind" in perlfunc*.

binmode() on closed filehandle %s

(W unopened) You tried binmode() on a filehandle that was never opened. Check you control flow and number of arguments.

Bit vector size > 32 non-portable

(W portable) Using bit vector sizes larger than 32 is non-portable.

Bizarre copy of %s in %s

(P) Perl detected an attempt to copy an internal value that is not copyable.

Buffer overflow in prime_env_iter: %s

(W internal) A warning peculiar to VMS. While Perl was preparing to iterate over %ENV, it encountered a logical name or symbol definition which was too long, so it was truncated to the string shown.

Callback called exit

(F) A subroutine invoked from an external package via call_sv() exited by calling exit.

%s() called too early to check prototype

(W prototype) You've called a function that has a prototype before the parser saw a definition or declaration for it, and Perl could not check that the call conforms to the prototype. You need to either add an early prototype declaration for the subroutine in question, or move the subroutine definition ahead of the call to get proper prototype checking. Alternatively, if you are certain that you're calling the function correctly, you may put an ampersand before the name to avoid the warning. See *perlsub*.

Cannot compress integer in pack

(F) An argument to pack("w",...) was too large to compress. The BER compressed integer format can only be used with positive integers, and you attempted to compress Infinity or a very large number (> 1e308). See *"pack" in perlfunc*.

Cannot compress negative numbers in pack

(F) An argument to pack("w",...) was negative. The BER compressed integer format can only be used with positive integers. See *"pack" in perlfunc*.

Cannot convert a reference to %s to typeglob

(F) You manipulated Perl's symbol table directly, stored a reference in it, then tried to access that symbol via conventional Perl syntax. The access triggers Perl to autovivify that typeglob, but it there is no legal conversion from that type of reference to a typeglob.

Cannot copy to %s in %s

(P) Perl detected an attempt to copy a value to an internal type that cannot be directly assigned not.

Can only compress unsigned integers in pack

(F) An argument to pack("w",...) was not an integer. The BER compressed integer format can only be used with positive integers, and you attempted to compress something else. See *"pack" in perlfunc*.

Can't bless non-reference value

(F) Only hard references may be blessed. This is how Perl "enforces" encapsulation of objects. See *perlobj*.

Can't "break" in a loop topicalizer

(F) You called `break`, but you're in a `foreach` block rather than a `given` block. You probably meant to use `next` or `last`.

Can't "break" outside a given block

> (F) You called `break`, but you're not inside a `given` block.

Can't call method "%s" in empty package "%s"

> (F) You called a method correctly, and it correctly indicated a package functioning as a class, but that package doesn't have ANYTHING defined in it, let alone methods. See *perlobj*.

Can't call method "%s" on an undefined value

> (F) You used the syntax of a method call, but the slot filled by the object reference or package name contains an undefined value. Something like this will reproduce the error:

>     $BADREF = undef;
>     process $BADREF 1,2,3;
>     $BADREF->process(1,2,3);

Can't call method "%s" on unblessed reference

> (F) A method call must know in what package it's supposed to run. It ordinarily finds this out from the object reference you supply, but you didn't supply an object reference in this case. A reference isn't an object reference until it has been blessed. See *perlobj*.

Can't call method "%s" without a package or object reference

> (F) You used the syntax of a method call, but the slot filled by the object reference or package name contains an expression that returns a defined value which is neither an object reference nor a package name. Something like this will reproduce the error:

>     $BADREF = 42;
>     process $BADREF 1,2,3;
>     $BADREF->process(1,2,3);

Can't chdir to %s

> (F) You called `perl -x/foo/bar`, but `/foo/bar` is not a directory that you can chdir to, possibly because it doesn't exist.

Can't check filesystem of script "%s" for nosuid

> (P) For some reason you can't check the filesystem of the script for nosuid.

Can't coerce array into hash

> (F) You used an array where a hash was expected, but the array has no information on how to map from keys to array indices. You can do that only with arrays that have a hash reference at index 0.

Can't coerce %s to integer in %s

> (F) Certain types of SVs, in particular real symbol table entries (typeglobs), can't be forced to stop being what they are. So you can't say things like:

>     *foo += 1;

> You CAN say

>     $foo = *foo;
>     $foo += 1;

> but then $foo no longer contains a glob.

Can't coerce %s to number in %s

> (F) Certain types of SVs, in particular real symbol table entries (typeglobs), can't be forced to stop being what they are.

Can't coerce %s to string in %s

>(F) Certain types of SVs, in particular real symbol table entries (typeglobs), can't be forced to stop being what they are.

Can't "continue" outside a when block

>(F) You called `continue`, but you're not inside a `when` or `default` block.

Can't create pipe mailbox

>(P) An error peculiar to VMS. The process is suffering from exhausted quotas or other plumbing problems.

Can't declare class for non-scalar %s in "%s"

>(F) Currently, only scalar variables can be declared with a specific class qualifier in a "my", "our" or "state" declaration. The semantics may be extended for other types of variables in future.

Can't declare %s in "%s"

>(F) Only scalar, array, and hash variables may be declared as "my", "our" or "state" variables. They must have ordinary identifiers as names.

Can't do inplace edit: %s is not a regular file

>(S inplace) You tried to use the **-i** switch on a special file, such as a file in /dev, or a FIFO. The file was ignored.

Can't do inplace edit on %s: %s

>(S inplace) The creation of the new file failed for the indicated reason.

Can't do inplace edit without backup

>(F) You're on a system such as MS-DOS that gets confused if you try reading from a deleted (but still opened) file. You have to say `-i.bak`, or some such.

Can't do inplace edit: %s would not be unique

>(S inplace) Your filesystem does not support filenames longer than 14 characters and Perl was unable to create a unique filename during inplace editing with the **-i** switch. The file was ignored.

Can't do {n,m} with n > m in regex; marked by <-- HERE in m/%s/

>(F) Minima must be less than or equal to maxima. If you really want your regexp to match something 0 times, just put {0}. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Can't do setegid!

>(P) The setegid() call failed for some reason in the setuid emulator of suidperl.

Can't do seteuid!

>(P) The setuid emulator of suidperl failed for some reason.

Can't do setuid

>(F) This typically means that ordinary perl tried to exec suidperl to do setuid emulation, but couldn't exec it. It looks for a name of the form sperl5.000 in the same directory that the perl executable resides under the name perl5.000, typically /usr/local/bin on Unix machines. If the file is there, check the execute permissions. If it isn't, ask your sysadmin why he and/or she removed it.

Can't do waitpid with flags

>(F) This machine doesn't have either waitpid() or wait4(), so only waitpid() without flags is

Can't emulate%s on #! line

> (F) The #! line specifies a switch that doesn't make sense at this point. For example, it'd be kind of silly to put a **-x** on the #! line.

Can't %s %s-endian %ss on this platform

> (F) Your platform's byte-order is neither big-endian nor little-endian, or it has a very strange pointer size. Packing and unpacking big- or little-endian floating point values and pointers may not be possible. See *"pack" in perlfunc*.

Can't exec "%s": %s

> (W exec) A system(), exec(), or piped open call could not execute the named program for the indicated reason. Typical reasons include: the permissions were wrong on the file, the file wasn't found in $ENV{PATH}, the executable in question was compiled for another architecture, or the #! line in a script points to an interpreter that can't be run for similar reasons. (Or maybe your system doesn't support #! at all.)

Can't exec %s

> (F) Perl was trying to execute the indicated program for you because that's what the #! line said. If that's not what you wanted, you may need to mention "perl" on the #! line somewhere.

Can't execute %s

> (F) You used the **-S** switch, but the copies of the script to execute found in the PATH did not have correct permissions.

Can't find an opnumber for "%s"

> (F) A string of a form CORE::word was given to prototype(), but there is no builtin with the name word.

Can't find %s character property "%s"

> (F) You used \p{} or \P{} but the character property by that name could not be found. Maybe you misspelled the name of the property (remember that the names of character properties consist only of alphanumeric characters), or maybe you forgot the Is or In prefix?

Can't find label %s

> (F) You said to goto a label that isn't mentioned anywhere that it's possible for us to go to. See *"goto" in perlfunc*.

Can't find %s on PATH

> (F) You used the **-S** switch, but the script to execute could not be found in the PATH.

Can't find %s on PATH, '.' not in PATH

> (F) You used the **-S** switch, but the script to execute could not be found in the PATH, or at least not with the correct permissions. The script exists in the current directory, but PATH prohibits running it.

Can't find string terminator %s anywhere before EOF

> (F) Perl strings can stretch over multiple lines. This message means that the closing delimiter was omitted. Because bracketed quotes count nesting levels, the following is missing its final parenthesis:
>
> ```
> print q(The character '(' starts a side comment.);
> ```
>
> If you're getting this error from a here-document, you may have included unseen whitespace before or after your closing tag. A good programmer's editor will have a way to help you find these characters.

Can't find Unicode property definition "%s"

(F) You may have tried to use `\p` which means a Unicode property (for example `\p{Lu}` is all uppercase letters). If you did mean to use a Unicode property, see *perlunicode* for the list of known properties. If you didn't mean to use a Unicode property, escape the `\p`, either by `\\p` (just the `\p`) or by `\Q\p` (the rest of the string, until possible `\E`).

Can't fork

> (F) A fatal error occurred while trying to fork while opening a pipeline.

Can't get filespec - stale stat buffer?

> (S) A warning peculiar to VMS. This arises because of the difference between access checks under VMS and under the Unix model Perl assumes. Under VMS, access checks are done by filename, rather than by bits in the stat buffer, so that ACLs and other protections can be taken into account. Unfortunately, Perl assumes that the stat buffer contains all the necessary information, and passes it, instead of the filespec, to the access checking routine. It will try to retrieve the filespec using the device name and FID present in the stat buffer, but this works only if you haven't made a subsequent call to the CRTL stat() routine, because the device name is overwritten with each call. If this warning appears, the name lookup failed, and the access checking routine gave up and returned FALSE, just to be conservative. (Note: The access checking routine knows about the Perl `stat` operator and file tests, so you shouldn't ever see this warning in response to a Perl command; it arises only if some internal code takes stat buffers lightly.)

Can't get pipe mailbox device name

> (P) An error peculiar to VMS. After creating a mailbox to act as a pipe, Perl can't retrieve its name for later use.

Can't get SYSGEN parameter value for MAXBUF

> (P) An error peculiar to VMS. Perl asked $GETSYI how big you want your mailbox buffers to be, and didn't get an answer.

Can't "goto" into the middle of a foreach loop

> (F) A "goto" statement was executed to jump into the middle of a foreach loop. You can't get there from here. See *"goto" in perlfunc*.

Can't "goto" out of a pseudo block

> (F) A "goto" statement was executed to jump out of what might look like a block, except that it isn't a proper block. This usually occurs if you tried to jump out of a sort() block or subroutine, which is a no-no. See *"goto" in perlfunc*.

Can't goto subroutine from a sort sub (or similar callback)

> (F) The "goto subroutine" call can't be used to jump out of the comparison sub for a sort(), or from a similar callback (such as the reduce() function in List::Util).

Can't goto subroutine from an eval-%s

> (F) The "goto subroutine" call can't be used to jump out of an eval "string" or block.

Can't goto subroutine outside a subroutine

> (F) The deeply magical "goto subroutine" call can only replace one subroutine call for another. It can't manufacture one out of whole cloth. In general you should be calling it out of only an AUTOLOAD routine anyway. See *"goto" in perlfunc*.

Can't ignore signal CHLD, forcing to default

> (W signal) Perl has detected that it is being run with the SIGCHLD signal (sometimes known as SIGCLD) disabled. Since disabling this signal will interfere with proper determination of exit status of child processes, Perl has reset the signal to its default value. This situation typically indicates that the parent program under which Perl may be running (e.g. cron) is being very

Can't "last" outside a loop block

> (F) A "last" statement was executed to break out of the current block, except that there's this itty bitty problem called there isn't a current block. Note that an "if" or "else" block doesn't count as a "loopish" block, as doesn't a block given to sort(), map() or grep(). You can usually double the curlies to get the same effect though, because the inner curlies will be considered a block that loops once. See *"last" in perlfunc*.

Can't linearize anonymous symbol table

> (F) Perl tried to calculate the method resolution order (MRO) of a package, but failed because the package stash has no name.

Can't load '%s' for module %s

> (F) The module you tried to load failed to load a dynamic extension. This may either mean that you upgraded your version of perl to one that is incompatible with your old dynamic extensions (which is known to happen between major versions of perl), or (more likely) that your dynamic extension was built against an older version of the library that is installed on your system. You may need to rebuild your old dynamic extensions.

Can't localize lexical variable %s

> (F) You used local on a variable name that was previously declared as a lexical variable using "my" or "state". This is not allowed. If you want to localize a package variable of the same name, qualify it with the package name.

Can't localize through a reference

> (F) You said something like `local $$ref`, which Perl can't currently handle, because when it goes to restore the old value of whatever $ref pointed to after the scope of the local() is finished, it can't be sure that $ref will still be a reference.

Can't locate %s

> (F) You said to `do` (or `require`, or `use`) a file that couldn't be found. Perl looks for the file in all the locations mentioned in @INC, unless the file name included the full path to the file. Perhaps you need to set the PERL5LIB or PERL5OPT environment variable to say where the extra library is, or maybe the script needs to add the library name to @INC. Or maybe you just misspelled the name of the file. See *"require" in perlfunc* and *lib*.

Can't locate auto/%s.al in @INC

> (F) A function (or method) was called in a package which allows autoload, but there is no function to autoload. Most probable causes are a misprint in a function/method name or a failure to `AutoSplit` the file, say, by doing `make install`.

Can't locate loadable object for module %s in @INC

> (F) The module you loaded is trying to load an external library, like for example, `foo.so` or `bar.dll`, but the *DynaLoader* module was unable to locate this library. See *DynaLoader*.

Can't locate object method "%s" via package "%s"

> (F) You called a method correctly, and it correctly indicated a package functioning as a class, but that package doesn't define that particular method, nor does any of its base classes. See *perlobj*.

Can't locate package %s for @%s::ISA

> (W syntax) The @ISA array contained the name of another package that doesn't seem to exist.

Can't locate PerlIO%s

> (F) You tried to use in open() a PerlIO layer that does not exist, e.g. open(FH, ">:nosuchlayer", "somefile").

Can't make list assignment to \%ENV on this system

> (F) List assignment to %ENV is not supported on some systems, notably VMS.

Can't modify %s in %s

> (F) You aren't allowed to assign to the item indicated, or otherwise try to change it, such as with an auto-increment.

Can't modify nonexistent substring

> (P) The internal routine that does assignment to a substr() was handed a NULL.

Can't modify non-lvalue subroutine call

> (F) Subroutines meant to be used in lvalue context should be declared as such, see *"Lvalue subroutines" in perlsub*.

Can't msgrcv to read-only var

> (F) The target of a msgrcv must be modifiable to be used as a receive buffer.

Can't "next" outside a loop block

> (F) A "next" statement was executed to reiterate the current block, but there isn't a current block. Note that an "if" or "else" block doesn't count as a "loopish" block, as doesn't a block given to sort(), map() or grep(). You can usually double the curlies to get the same effect though, because the inner curlies will be considered a block that loops once. See *"next" in perlfunc*.

Can't open %s: %s

> (S inplace) The implicit opening of a file through use of the <> filehandle, either implicitly under the −n or −p command-line switches, or explicitly, failed for the indicated reason. Usually this is because you don't have read permission for a file which you named on the command line.

Can't open a reference

> (W io) You tried to open a scalar reference for reading or writing, using the 3-arg open() syntax :
>
> ```
>     open FH, '>', $ref;
> ```
>
> but your version of perl is compiled without perlio, and this form of open is not supported.

Can't open bidirectional pipe

> (W pipe) You tried to say open(CMD, "|cmd|"), which is not supported. You can try any of several modules in the Perl library to do this, such as IPC::Open2. Alternately, direct the pipe's output to a file using ">", and then read it in under a different file handle.

Can't open error file %s as stderr

> (F) An error peculiar to VMS. Perl does its own command line redirection, and couldn't open the file specified after '2>' or '2>>' on the command line for writing.

Can't open input file %s as stdin

> (F) An error peculiar to VMS. Perl does its own command line redirection, and couldn't open the file specified after '<' on the command line for reading.

Can't open output file %s as stdout

> (F) An error peculiar to VMS. Perl does its own command line redirection, and couldn't open the file specified after '>' or '>>' on the command line for writing.

Can't open output pipe (name: %s)

---

(P) An error peculiar to VMS. Perl does its own command line redirection, and couldn't open the pipe into which to send data destined for stdout.

Can't open perl script%s

(F) The script you specified can't be opened for the indicated reason.

If you're debugging a script that uses #!, and normally relies on the shell's $PATH search, the -S option causes perl to do that search, so you don't have to type the path or `which $scriptname`.

Can't read CRTL environ

(S) A warning peculiar to VMS. Perl tried to read an element of %ENV from the CRTL's internal environment array and discovered the array was missing. You need to figure out where your CRTL misplaced its environ or define *PERL_ENV_TABLES* (see *perlvms*) so that environ is not searched.

Can't "redo" outside a loop block

(F) A "redo" statement was executed to restart the current block, but there isn't a current block. Note that an "if" or "else" block doesn't count as a "loopish" block, as doesn't a block given to sort(), map() or grep(). You can usually double the curlies to get the same effect though, because the inner curlies will be considered a block that loops once. See *"redo" in perlfunc*.

Can't remove %s: %s, skipping file

(S inplace) You requested an inplace edit without creating a backup file. Perl was unable to remove the original file to replace it with the modified file. The file was left unmodified.

Can't rename %s to %s: %s, skipping file

(S inplace) The rename done by the **-i** switch failed for some reason, probably because you don't have write permission to the directory.

Can't reopen input pipe (name: %s) in binary mode

(P) An error peculiar to VMS. Perl thought stdin was a pipe, and tried to reopen it to accept binary data. Alas, it failed.

Can't resolve method `%s' overloading `%s' in package `%s'

(F|P) Error resolving overloading specified by a method name (as opposed to a subroutine reference): no such method callable via the package. If method name is ???, this is an internal error.

Can't reswap uid and euid

(P) The setreuid() call failed for some reason in the setuid emulator of suidperl.

Can't return %s from lvalue subroutine

(F) Perl detected an attempt to return illegal lvalues (such as temporary or readonly values) from a subroutine used as an lvalue. This is not allowed.

Can't return outside a subroutine

(F) The return statement was executed in mainline code, that is, where there was no subroutine call to return out of. See *perlsub*.

Can't return %s to lvalue scalar context

(F) You tried to return a complete array or hash from an lvalue subroutine, but you called the subroutine in a way that made Perl think you meant to return only one value. You probably meant to write parentheses around the call to the subroutine, which tell Perl that the call should be in list context.

Can't stat script "%s"

>   (P) For some reason you can't fstat() the script even though you have it open already. Bizarre.

Can't swap uid and euid

>   (P) The setreuid() call failed for some reason in the setuid emulator of suidperl.

Can't take log of %g

>   (F) For ordinary real numbers, you can't take the logarithm of a negative number or zero. There's a Math::Complex package that comes standard with Perl, though, if you really want to do that for the negative numbers.

Can't take sqrt of %g

>   (F) For ordinary real numbers, you can't take the square root of a negative number. There's a Math::Complex package that comes standard with Perl, though, if you really want to do that.

Can't undef active subroutine

>   (F) You can't undefine a routine that's currently running. You can, however, redefine it while it's running, and you can even undef the redefined subroutine while the old routine is running. Go figure.

Can't unshift

>   (F) You tried to unshift an "unreal" array that can't be unshifted, such as the main Perl stack.

Can't upgrade %s (%d) to %d

>   (P) The internal sv_upgrade routine adds "members" to an SV, making it into a more specialized kind of SV. The top several SV types are so specialized, however, that they cannot be interconverted. This message indicates that such a conversion was attempted.

Can't use anonymous symbol table for method lookup

>   (F) The internal routine that does method lookup was handed a symbol table that doesn't have a name. Symbol tables can become anonymous for example by undefining stashes: `undef %Some::Package::`.

Can't use an undefined value as %s reference

>   (F) A value used as either a hard reference or a symbolic reference must be a defined value. This helps to delurk some insidious errors.

Can't use bareword ("%s") as %s ref while "strict refs" in use

>   (F) Only hard references are allowed by "strict refs". Symbolic references are disallowed. See *perlref*.

Can't use %! because Errno.pm is not available

>   (F) The first time the %! hash is used, perl automatically loads the Errno.pm module. The Errno module is expected to tie the %! hash to provide symbolic names for `$!` errno values.

Can't use both '<' and '>' after type '%c' in %s

>   (F) A type cannot be forced to have both big-endian and little-endian byte-order at the same time, so this combination of modifiers is not allowed. See *"pack" in perlfunc*.

Can't use %s for loop variable

>   (F) Only a simple scalar variable may be used as a loop variable on a foreach.

Can't use global %s in "%s"

>   (F) You tried to declare a magical variable as a lexical variable. This is not allowed, because the magic can be tied to only one location (namely the global variable) and it would be incredibly confusing to have variables in your program that looked like magical variables but

Can't use %s't in a group with different byte-order in %s

> (F) You attempted to force a different byte-order on a type that is already inside a group with a byte-order modifier. For example you cannot force little-endianness on a type that is inside a big-endian group.

Can't use "my %s" in sort comparison

> (F) The global variables $a and $b are reserved for sort comparisons. You mentioned $a or $b in the same line as the <=> or cmp operator, and the variable had earlier been declared as a lexical variable. Either qualify the sort variable with the package name, or rename the lexical variable.

Can't use %s ref as %s ref

> (F) You've mixed up your reference types. You have to dereference a reference of the type needed. You can use the ref() function to test the type of the reference, if need be.

Can't use string ("%s") as %s ref while "strict refs" in use

> (F) Only hard references are allowed by "strict refs". Symbolic references are disallowed. See *perlref*.

Can't use subscript on %s

> (F) The compiler tried to interpret a bracketed expression as a subscript. But to the left of the brackets was an expression that didn't look like a hash or array reference, or anything else subscriptable.

Can't use \%c to mean $%c in expression

> (W syntax) In an ordinary expression, backslash is a unary operator that creates a reference to its argument. The use of backslash to indicate a backreference to a matched substring is valid only as part of a regular expression pattern. Trying to do this in ordinary Perl code produces a value that prints out looking like SCALAR(0xdecaf). Use the $1 form instead.

Can't use "when" outside a topicalizer

> (F) You have used a when() block that is neither inside a `foreach` loop nor a `given` block. (Note that this error is issued on exit from the `when` block, so you won't get the error if the match fails, or if you use an explicit `continue`.)

Can't weaken a nonreference

> (F) You attempted to weaken something that was not a reference. Only references can be weakened.

Can't x= to read-only value

> (F) You tried to repeat a constant value (often the undefined value) with an assignment operator, which implies modifying the value itself. Perhaps you need to copy the value to a temporary, and repeat that.

Character in 'C' format wrapped in pack

> (W pack) You said
>
> ```
> pack("C", $x)
> ```
>
> where $x is either less than 0 or more than 255; the `"C"` format is only for encoding native operating system characters (ASCII, EBCDIC, and so on) and not for Unicode characters, so Perl behaved as if you meant
>
> ```
> pack("C", $x & 255)
> ```
>
> If you actually want to pack Unicode codepoints, use the `"U"` format instead.

Character in 'W' format wrapped in pack

> (W pack) You said
>
> ```
> pack("U0W", $x)
> ```
>
> where $x is either less than 0 or more than 255. However, U0-mode expects all values to fall in the interval [0, 255], so Perl behaved as if you meant:
>
> ```
> pack("U0W", $x & 255)
> ```

Character in 'c' format wrapped in pack

> (W pack) You said
>
> ```
> pack("c", $x)
> ```
>
> where $x is either less than -128 or more than 127; the "c" format is only for encoding native operating system characters (ASCII, EBCDIC, and so on) and not for Unicode characters, so Perl behaved as if you meant
>
> ```
> pack("c", $x & 255);
> ```
>
> If you actually want to pack Unicode codepoints, use the "U" format instead.

Character in '%c' format wrapped in unpack

> (W unpack) You tried something like
>
> ```
> unpack("H", "\x{2a1}")
> ```
>
> where the format expects to process a byte (a character with a value below 256), but a higher value was provided instead. Perl uses the value modulus 256 instead, as if you had provided:
>
> ```
> unpack("H", "\x{a1}")
> ```

Character(s) in '%c' format wrapped in pack

> (W pack) You tried something like
>
> ```
> pack("u", "\x{1f3}b")
> ```
>
> where the format expects to process a sequence of bytes (character with a value below 256), but some of the characters had a higher value. Perl uses the character values modulus 256 instead, as if you had provided:
>
> ```
> pack("u", "\x{f3}b")
> ```

Character(s) in '%c' format wrapped in unpack

> (W unpack) You tried something like
>
> ```
> unpack("s", "\x{1f3}b")
> ```
>
> where the format expects to process a sequence of bytes (character with a value below 256), but some of the characters had a higher value. Perl uses the character values modulus 256 instead, as if you had provided:
>
> ```
> unpack("s", "\x{f3}b")
> ```

close() on unopened filehandle %s

> (W unopened) You tried to close a filehandle that was never opened.

closedir() attempted on invalid dirhandle %s

> (W io) The dirhandle you tried to close is either closed or not really a dirhandle. Check your

control flow.

Code missing after '/'

(F) You had a (sub-)template that ends with a '/'. There must be another template code following the slash. See *"pack" in perlfunc*.

%s: Command not found

(A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

Compilation failed in require

(F) Perl could not compile a file specified in a `require` statement. Perl uses this generic message when none of the errors that it encountered were severe enough to halt compilation immediately.

Complex regular subexpression recursion limit (%d) exceeded

(W regexp) The regular expression engine uses recursion in complex situations where back-tracking is required. Recursion depth is limited to 32766, or perhaps less in architectures where the stack cannot grow arbitrarily. ("Simple" and "medium" situations are handled without recursion and are not subject to a limit.) Try shortening the string under examination; looping in Perl code (e.g. with `while`) rather than in the regular expression engine; or rewriting the regular expression so that it is simpler or backtracks less. (See *perlfaq2* for information on *Mastering Regular Expressions*.)

cond_broadcast() called on unlocked variable

(W threads) Within a thread-enabled program, you tried to call cond_broadcast() on a variable which wasn't locked. The cond_broadcast() function is used to wake up another thread that is waiting in a cond_wait(). To ensure that the signal isn't sent before the other thread has a chance to enter the wait, it is usual for the signaling thread to first wait for a lock on variable. This lock attempt will only succeed after the other thread has entered cond_wait() and thus relinquished the lock.

cond_signal() called on unlocked variable

(W threads) Within a thread-enabled program, you tried to call cond_signal() on a variable which wasn't locked. The cond_signal() function is used to wake up another thread that is waiting in a cond_wait(). To ensure that the signal isn't sent before the other thread has a chance to enter the wait, it is usual for the signaling thread to first wait for a lock on variable. This lock attempt will only succeed after the other thread has entered cond_wait() and thus relinquished the lock.

connect() on closed socket %s

(W closed) You tried to do a connect on a closed socket. Did you forget to check the return value of your socket() call? See *"connect" in perlfunc*.

Constant(%s)%s: %s

(F) The parser found inconsistencies either while attempting to define an overloaded constant, or when trying to find the character name specified in the `\N{...}` escape. Perhaps you forgot to load the corresponding `overload` or `charnames` pragma? See *charnames* and *overload*.

Constant(%s)%s: %s in regex; marked by <-- HERE in m/%s/

(F) The parser found inconsistencies while attempting to find the character name specified in the `\N{...}` escape. Perhaps you forgot to load the corresponding `charnames` pragma? See *charnames*.

Constant is not %s reference

(F) A constant value (perhaps declared using the `use constant` pragma) is being dereferenced, but it amounts to the wrong type of reference. The message indicates the type of reference that was expected. This usually indicates a syntax error in dereferencing the constant value. See *"Constant Functions" in perlsub* and *constant*.

**Constant subroutine %s redefined**

(S) You redefined a subroutine which had previously been eligible for inlining. See *"Constant Functions" in perlsub* for commentary and workarounds.

**Constant subroutine %s undefined**

(W misc) You undefined a subroutine which had previously been eligible for inlining. See *"Constant Functions" in perlsub* for commentary and workarounds.

**Copy method did not return a reference**

(F) The method which overloads "=" is buggy. See *"Copy Constructor" in overload*.

**CORE::%s is not a keyword**

(F) The CORE:: namespace is reserved for Perl keywords.

**corrupted regexp pointers**

(P) The regular expression engine got confused by what the regular expression compiler gave it.

**corrupted regexp program**

(P) The regular expression engine got passed a regexp program without a valid magic number.

**Corrupt malloc ptr 0x%lx at 0x%lx**

(P) The malloc package that comes with Perl had an internal failure.

**Count after length/code in unpack**

(F) You had an unpack template indicating a counted-length string, but you have also specified an explicit size for the string. See *"pack" in perlfunc*.

**Deep recursion on subroutine "%s"**

(W recursion) This subroutine has called itself (directly or indirectly) 100 times more than it has returned. This probably indicates an infinite recursion, unless you're writing strange benchmark programs, in which case it indicates something else.

This threshold can be changed from 100, by recompiling the *perl* binary, setting the C pre-processor macro `PERL_SUB_DEPTH_WARN` to the desired value.

**defined(@array) is deprecated**

(D deprecated) defined() is not usually useful on arrays because it checks for an undefined *scalar* value. If you want to see if the array is empty, just use `if (@array) { # not empty }` for example.

**defined(%hash) is deprecated**

(D deprecated) defined() is not usually useful on hashes because it checks for an undefined *scalar* value. If you want to see if the hash is empty, just use `if (%hash) { # not empty }` for example.

**%s defines neither package nor VERSION--version check failed**

(F) You said something like "use Module 42" but in the Module file there are neither package declarations nor a `$VERSION`.

**Delimiter for here document is too long**

(F) In a here document construct like `<<FOO`, the label `FOO` is too long for Perl to handle. You have to be seriously twisted to write code that triggers this error.

Deprecated use of my() in false conditional

(D deprecated) You used a declaration similar to `my $x if 0`. There has been a long-standing bug in Perl that causes a lexical variable not to be cleared at scope exit when its declaration includes a false conditional. Some people have exploited this bug to achieve a kind of static variable. Since we intend to fix this bug, we don't want people relying on this behavior. You can achieve a similar static effect by declaring the variable in a separate block outside the function, eg

```
sub f { my $x if 0; return $x++ }
```

becomes

```
{ my $x; sub f { return $x++ } }
```

Beginning with perl 5.9.4, you can also use `state` variables to have lexicals that are initialized only once (see *feature*):

```
sub f { state $x; return $x++ }
```

DESTROY created new reference to dead object '%s'

(F) A DESTROY() method created a new reference to the object which is just being DESTROYed. Perl is confused, and prefers to abort rather than to create a dangling reference.

Did not produce a valid header

See Server error.

%s did not return a true value

(F) A required (or used) file must return a true value to indicate that it compiled correctly and ran its initialization code correctly. It's traditional to end such a file with a "1;", though any true value would do. See *"require" in perlfunc*.

(Did you mean &%s instead?)

(W) You probably referred to an imported subroutine &FOO as $FOO or some such.

(Did you mean "local" instead of "our"?)

(W misc) Remember that "our" does not localize the declared global variable. You have declared it again in the same lexical scope, which seems superfluous.

(Did you mean $ or @ instead of %?)

(W) You probably said %hash{$key} when you meant $hash{$key} or @hash{@keys}. On the other hand, maybe you just meant %hash and got carried away.

Died

(F) You passed die() an empty string (the equivalent of `die ""`) or you called it with no args and both `$@` and `$_` were empty.

Document contains no data

See Server error.

%s does not define %s::VERSION--version check failed

(F) You said something like "use Module 42" but the Module did not define a `$VERSION`.

'/' does not take a repeat count

(F) You cannot put a repeat count of any kind right after the '/' code. See *"pack" in perlfunc*.

Don't know how to handle magic of type '%s'

(P) The internal handling of magical variables has been cursed.

do_study: out of memory

(P) This should have been caught by safemalloc() instead.

(Do you need to predeclare %s?)

(S syntax) This is an educated guess made in conjunction with the message "%s found where operator expected". It often means a subroutine or module name is being referenced that hasn't been declared yet. This may be because of ordering problems in your file, or because of a missing "sub", "package", "require", or "use" statement. If you're referencing something that isn't defined yet, you don't actually have to define the subroutine or package before the current location. You can use an empty "sub foo;" or "package FOO;" to enter a "forward" declaration.

dump() better written as CORE::dump()

(W misc) You used the obsolescent `dump()` built-in function, without fully qualifying it as `CORE::dump()`. Maybe it's a typo. See *"dump" in perlfunc*.

dump is not supported

(F) Your machine doesn't support dump/undump.

Duplicate free() ignored

(S malloc) An internal routine called free() on something that had already been freed.

Duplicate modifier '%c' after '%c' in %s

(W) You have applied the same modifier more than once after a type in a pack template. See *"pack" in perlfunc*.

elseif should be elsif

(S syntax) There is no keyword "elseif" in Perl because Larry thinks it's ugly. Your code will be interpreted as an attempt to call a method named "elseif" for the class returned by the following block. This is unlikely to be what you want.

Empty %s

(F) `\p` and `\P` are used to introduce a named Unicode property, as described in *perlunicode* and *perlre*. You used `\p` or `\P` in a regular expression without specifying the property name.

entering effective %s failed

(F) While under the `use filetest` pragma, switching the real and effective uids or gids failed.

%ENV is aliased to %s

(F) You're running under taint mode, and the `%ENV` variable has been aliased to another hash, so it doesn't reflect anymore the state of the program's environment. This is potentially insecure.

Error converting file specification %s

(F) An error peculiar to VMS. Because Perl may have to deal with file specifications in either VMS or Unix syntax, it converts them to a single form when it must operate on them directly. Either you've passed an invalid file specification to Perl, or you've found a case the conversion routines don't handle. Drat.

%s: Eval-group in insecure regular expression

(F) Perl detected tainted data when trying to compile a regular expression that contains the `(?{ ... })` zero-width assertion, which is unsafe. See *"(?{ code })" in perlre*, and *perlsec*.

**%s: Eval-group not allowed at runtime, use re 'eval'**

(F) Perl tried to compile a regular expression containing the `(?{ ... })` zero-width assertion at run time, as it would when the pattern contains interpolated values. Since that is a security risk, it is not allowed. If you insist, you may still do this by explicitly building the pattern from an interpolated string at run time and using that in an eval(). See *"(?{ code })" in perlre*.

**%s: Eval-group not allowed, use re 'eval'**

(F) A regular expression contained the `(?{ ... })` zero-width assertion, but that construct is only allowed when the `use re 'eval'` pragma is in effect. See *"(?{ code })" in perlre*.

**EVAL without pos change exceeded limit in regex; marked by <-- HERE in m/%s/**

(F) You used a pattern that nested too many EVAL calls without consuming any text. Restructure the pattern so that text is consumed.

The <-- HERE shows in the regular expression about where the problem was discovered.

**Excessively long <> operator**

(F) The contents of a <> operator may not exceed the maximum size of a Perl identifier. If you're just trying to glob a long list of filenames, try using the glob() operator, or put the filenames into a variable and glob that.

**exec? I'm not \*that\* kind of operating system**

(F) The `exec` function is not implemented in MacPerl. See *perlport*.

**Execution of %s aborted due to compilation errors**

(F) The final summary message when a Perl compilation fails.

**Exiting eval via %s**

(W exiting) You are exiting an eval by unconventional means, such as a goto, or a loop control statement.

**Exiting format via %s**

(W exiting) You are exiting a format by unconventional means, such as a goto, or a loop control statement.

**Exiting pseudo-block via %s**

(W exiting) You are exiting a rather special block construct (like a sort block or subroutine) by unconventional means, such as a goto, or a loop control statement. See *"sort" in perlfunc*.

**Exiting subroutine via %s**

(W exiting) You are exiting a subroutine by unconventional means, such as a goto, or a loop control statement.

**Exiting substitution via %s**

(W exiting) You are exiting a substitution by unconventional means, such as a return, a goto, or a loop control statement.

**Explicit blessing to '' (assuming package main)**

(W misc) You are blessing a reference to a zero length string. This has the effect of blessing the reference into the package main. This is usually not what you want. Consider providing a default target package, e.g. bless($ref, $p || 'MyPackage');

**%s: Expression syntax**

(A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or

manually feed your script into Perl yourself.

%s failed--call queue aborted

(F) An untrapped exception was raised while executing a UNITCHECK, CHECK, INIT, or END subroutine. Processing of the remainder of the queue of such routines has been prematurely ended.

False [] range "%s" in regex; marked by <-- HERE in m/%s/

(W regexp) A character class range must start and end at a literal character, not another character class like \d or [:alpha:]. The "-" in your false range is interpreted as a literal "-". Consider quoting the "-", "\-". The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Fatal VMS error at %s, line %d

(P) An error peculiar to VMS. Something untoward happened in a VMS system service or RTL routine; Perl's exit status should provide more details. The filename in "at %s" and the line number in "line %d" tell you which section of the Perl source code is distressed.

fcntl is not implemented

(F) Your machine apparently doesn't implement fcntl(). What is this, a PDP-11 or something?

FETCHSIZE returned a negative value

(F) A tied array claimed to have a negative number of elements, which is not possible.

Field too wide in 'u' format in pack

(W pack) Each line in an uuencoded string start with a length indicator which can't encode values above 63. So there is no point in asking for a line length bigger than that. Perl will behave as if you specified u63 as format.

Filehandle %s opened only for input

(W io) You tried to write on a read-only filehandle. If you intended it to be a read-write filehandle, you needed to open it with "+<" or "+>" or "+>>" instead of with "<" or nothing. If you intended only to write the file, use ">" or ">>". See *"open" in perlfunc*.

Filehandle %s opened only for output

(W io) You tried to read from a filehandle opened only for writing, If you intended it to be a read/write filehandle, you needed to open it with "+<" or "+>" or "+>>" instead of with "<" or nothing. If you intended only to read from the file, use "<". See *"open" in perlfunc*. Another possibility is that you attempted to open filedescriptor 0 (also known as STDIN) for output (maybe you closed STDIN earlier?).

Filehandle %s reopened as %s only for input

(W io) You opened for reading a filehandle that got the same filehandle id as STDOUT or STDERR. This occurred because you closed STDOUT or STDERR previously.

Filehandle STDIN reopened as %s only for output

(W io) You opened for writing a filehandle that got the same filehandle id as STDIN. This occurred because you closed STDIN previously.

Final $ should be \$ or $name

(F) You must now decide whether the final $ in a string was meant to be a literal dollar sign, or was meant to introduce a variable name that happens to be missing. So you have to put either the backslash or the name.

flock() on closed filehandle %s

(W closed) The filehandle you're attempting to flock() got itself closed some time before now.

Check your control flow. flock() operates on filehandles. Are you attempting to call flock() on a dirhandle by the same name?

Format not terminated

(F) A format must be terminated by a line with a solitary dot. Perl got to the end of your file without finding such a line.

Format %s redefined

(W redefine) You redefined a format. To suppress this warning, say

```
    {
 no warnings 'redefine';
 eval "format NAME =...";
    }
```

Found = in conditional, should be ==

(W syntax) You said

```
    if ($foo = 123)
```

when you meant

```
    if ($foo == 123)
```

(or something like that).

%s found where operator expected

(S syntax) The Perl lexer knows whether to expect a term or an operator. If it sees what it knows to be a term when it was expecting to see an operator, it gives you this warning. Usually it indicates that an operator or delimiter was omitted, such as a semicolon.

gdbm store returned %d, errno %d, key "%s"

(S) A warning from the GDBM_File extension that a store failed.

gethostent not implemented

(F) Your C library apparently doesn't implement gethostent(), probably because if it did, it'd feel morally obligated to return every hostname on the Internet.

get%sname() on closed socket %s

(W closed) You tried to get a socket or peer socket name on a closed socket. Did you forget to check the return value of your socket() call?

getpwnam returned invalid UIC %#o for user "%s"

(S) A warning peculiar to VMS. The call to sys$getuai underlying the getpwnam operator returned an invalid UIC.

getsockopt() on closed socket %s

(W closed) You tried to get a socket option on a closed socket. Did you forget to check the return value of your socket() call? See *"getsockopt" in perlfunc*.

Global symbol "%s" requires explicit package name

(F) You've said "use strict" or "use strict vars", which indicates that all variables must either be lexically scoped (using "my" or "state"), declared beforehand using "our", or explicitly qualified to say which package the global variable is in (using "::").

glob failed (%s)

(W glob) Something went wrong with the external program(s) used for glob and <*.c>.

Usually, this means that you supplied a `glob` pattern that caused the external program to fail and exit with a nonzero status. If the message indicates that the abnormal exit resulted in a coredump, this may also mean that your csh (C shell) is broken. If so, you should change all of the csh-related variables in config.sh: If you have tcsh, make the variables refer to it as if it were csh (e.g. `full_csh='/usr/bin/tcsh'`); otherwise, make them all empty (except that `d_csh` should be `'undef'`) so that Perl will think csh is missing. In either case, after editing config.sh, run `./Configure -S` and rebuild Perl.

Glob not terminated

(F) The lexer saw a left angle bracket in a place where it was expecting a term, so it's looking for the corresponding right angle bracket, and not finding it. Chances are you left some needed parentheses out earlier in the line, and you really meant a "less than".

Got an error from DosAllocMem

(P) An error peculiar to OS/2. Most probably you're using an obsolete version of Perl, and this should not happen anyway.

goto must have label

(F) Unlike with "next" or "last", you're not allowed to goto an unspecified destination. See *"goto" in perlfunc*.

()-group starts with a count

(F) A ()-group started with a count. A count is supposed to follow something: a template character or a ()-group. See *"pack" in perlfunc*.

%s had compilation errors

(F) The final summary message when a `perl -c` fails.

Had to create %s unexpectedly

(S internal) A routine asked for a symbol from a symbol table that ought to have existed already, but for some reason it didn't, and had to be created on an emergency basis to prevent a core dump.

Hash %%s missing the % in argument %d of %s()

(D deprecated) Really old Perl let you omit the % on hash names in some spots. This is now heavily deprecated.

%s has too many errors

(F) The parser has given up trying to parse the program after 10 errors. Further error messages would likely be uninformative.

Hexadecimal number > 0xffffffff non-portable

(W portable) The hexadecimal number you specified is larger than 2**32-1 (4294967295) and therefore non-portable between systems. See *perlport* for more on portability concerns.

Identifier too long

(F) Perl limits identifiers (names for variables, functions, etc.) to about 250 characters for simple names, and somewhat more for compound names (like `$A::B`). You've exceeded Perl's limits. Future versions of Perl are likely to eliminate these arbitrary limitations.

Ignoring %s in character class in regex; marked by <-- HERE in m/%s/

(W) Named Unicode character escapes (\N{...}) may return multi-char or zero length sequences. When such an escape is used in a character class its behaviour is not well defined. Check that the correct escape has been used, and the correct charname handler is in scope.

Illegal binary digit %s

> (F) You used a digit other than 0 or 1 in a binary number.

Illegal binary digit %s ignored

> (W digit) You may have tried to use a digit other than 0 or 1 in a binary number. Interpretation of the binary number stopped before the offending digit.

Illegal character %s (carriage return)

> (F) Perl normally treats carriage returns in the program text as it would any other whitespace, which means you should never see this error when Perl was built using standard options. For some reason, your version of Perl appears to have been built without this support. Talk to your Perl administrator.

Illegal character in prototype for %s : %s

> (W syntax) An illegal character was found in a prototype declaration. Legal characters in prototypes are $, @, %, *, ;, [, ], &, and \.

Illegal declaration of anonymous subroutine

> (F) When using the sub keyword to construct an anonymous subroutine, you must always specify a block of code. See *perlsub*.

Illegal declaration of subroutine %s

> (F) A subroutine was not declared correctly. See *perlsub*.

Illegal division by zero

> (F) You tried to divide a number by 0. Either something was wrong in your logic, or you need to put a conditional in to guard against meaningless input.

Illegal hexadecimal digit %s ignored

> (W digit) You may have tried to use a character other than 0 - 9 or A - F, a - f in a hexadecimal number. Interpretation of the hexadecimal number stopped before the illegal character.

Illegal modulus zero

> (F) You tried to divide a number by 0 to get the remainder. Most numbers don't take to this kindly.

Illegal number of bits in vec

> (F) The number of bits in vec() (the third argument) must be a power of two from 1 to 32 (or 64, if your platform supports that).

Illegal octal digit %s

> (F) You used an 8 or 9 in an octal number.

Illegal octal digit %s ignored

> (W digit) You may have tried to use an 8 or 9 in an octal number. Interpretation of the octal number stopped before the 8 or 9.

Illegal switch in PERL5OPT: %s

> (X) The PERL5OPT environment variable may only be used to set the following switches: **-[CDIMUdmtw]**.

Ill-formed CRTL environ value "%s"

> (W internal) A warning peculiar to VMS. Perl tried to read the CRTL's internal environ array, and encountered an element without the = delimiter used to separate keys from values. The element is ignored.

Ill-formed message in prime_env_iter: |%s|

> (W internal) A warning peculiar to VMS. Perl tried to read a logical name or CLI symbol definition when preparing to iterate over %ENV, and didn't see the expected delimiter between key and value, so the line was ignored.

(in cleanup) %s

> (W misc) This prefix usually indicates that a DESTROY() method raised the indicated exception. Since destructors are usually called by the system at arbitrary points during execution, and often a vast number of times, the warning is issued only once for any number of failures that would otherwise result in the same message being repeated.
>
> Failure of user callbacks dispatched using the `G_KEEPERR` flag could also result in this warning. See *"G_KEEPERR" in perlcall*.

Inconsistent hierarchy during C3 merge of class '%s': merging failed on parent '%s'

> (F) The method resolution order (MRO) of the given class is not C3-consistent, and you have enabled the C3 MRO for this class. See the C3 documentation in *mro* for more information.

In EBCDIC the v-string components cannot exceed 2147483647

> (F) An error peculiar to EBCDIC. Internally, v-strings are stored as Unicode code points, and encoded in EBCDIC as UTF-EBCDIC. The UTF-EBCDIC encoding is limited to code points no larger than 2147483647 (0x7FFFFFFF).

Infinite recursion in regex; marked by <-- HERE in m/%s/

> (F) You used a pattern that references itself without consuming any input text. You should check the pattern to ensure that recursive patterns either consume text or fail.
>
> The <-- HERE shows in the regular expression about where the problem was discovered.

Initialization of state variables in list context currently forbidden

> (F) Currently the implementation of "state" only permits the initialization of scalar variables in scalar context. Re-write `state ($a) = 42` as `state $a = 42` to change from list to scalar context. Constructions such as `state (@a) = foo()` will be supported in a future perl release.

Insecure dependency in %s

> (F) You tried to do something that the tainting mechanism didn't like. The tainting mechanism is turned on when you're running setuid or setgid, or when you specify **-T** to turn it on explicitly. The tainting mechanism labels all data that's derived directly or indirectly from the user, who is considered to be unworthy of your trust. If any such data is used in a "dangerous" operation, you get this error. See *perlsec* for more information.

Insecure directory in %s

> (F) You can't use system(), exec(), or a piped open in a setuid or setgid script if `$ENV{PATH}` contains a directory that is writable by the world. Also, the PATH must not contain any relative directory. See *perlsec*.

Insecure $ENV{%s} while running %s

> (F) You can't use system(), exec(), or a piped open in a setuid or setgid script if any of `$ENV{PATH}`, `$ENV{IFS}`, `$ENV{CDPATH}`, `$ENV{ENV}`, `$ENV{BASH_ENV}` or `$ENV{TERM}` are derived from data supplied (or potentially supplied) by the user. The script must set the path to a known value, using trustworthy data. See *perlsec*.

Integer overflow in %s number

> (W overflow) The hexadecimal, octal or binary number you have specified either as a literal or as an argument to hex() or oct() is too big for your architecture, and has been converted to a floating point number. On a 32-bit architecture the largest hexadecimal, octal or binary number

representable without overflow is 0xFFFFFFFF, 037777777777, or
0b11111111111111111111111111111111 respectively. Note that Perl transparently promotes
all numbers to a floating point representation internally--subject to loss of precision errors in
subsequent operations.

Integer overflow in format string for %s

(F) The indexes and widths specified in the format string of `printf()` or `sprintf()` are too
large. The numbers must not overflow the size of integers for your architecture.

Integer overflow in version

(F) Some portion of a version initialization is too large for the size of integers for your
architecture. This is not a warning because there is no rational reason for a version to try and
use a element larger than typically 2**32. This is usually caused by trying to use some odd
mathematical operation as a version, like 100/9.

Internal disaster in regex; marked by <-- HERE in m/%s/

(P) Something went badly wrong in the regular expression parser. The <-- HERE shows in the
regular expression about where the problem was discovered.

Internal inconsistency in tracking vforks

(S) A warning peculiar to VMS. Perl keeps track of the number of times you've called `fork`
and `exec`, to determine whether the current call to `exec` should affect the current script or a
subprocess (see *"exec LIST" in perlvms*). Somehow, this count has become scrambled, so
Perl is making a guess and treating this `exec` as a request to terminate the Perl script and
execute the specified command.

Internal urp in regex; marked by <-- HERE in m/%s/

(P) Something went badly awry in the regular expression parser. The <-- HERE shows in the
regular expression about where the problem was discovered.

%s (...) interpreted as function

(W syntax) You've run afoul of the rule that says that any list operator followed by parentheses
turns into a function, with all the list operators arguments found inside the parentheses. See
*"Terms and List Operators (Leftward)" in perlop*.

Invalid %s attribute: %s

The indicated attribute for a subroutine or variable was not recognized by Perl or by a
user-supplied handler. See *attributes*.

Invalid %s attributes: %s

The indicated attributes for a subroutine or variable were not recognized by Perl or by a
user-supplied handler. See *attributes*.

Invalid conversion in %s: "%s"

(W printf) Perl does not understand the given format conversion. See *"sprintf" in perlfunc*.

Invalid escape in the specified encoding in regex; marked by <-- HERE in m/%s/

(W regexp) The numeric escape (for example `\xHH`) of value < 256 didn't correspond to a
single character through the conversion from the encoding specified by the encoding pragma.
The escape was replaced with REPLACEMENT CHARACTER (U+FFFD) instead. The <--
HERE shows in the regular expression about where the escape was discovered.

Invalid mro name: '%s'

(F) You tried to `mro::set_mro("classname", "foo")` or `use mro 'foo'`, where `foo` is
not a valid method resolution order (MRO). (Currently, the only valid ones are `dfs` and `c3`).
See *mro*.

Invalid [] range "%s" in regex; marked by <-- HERE in m/%s/

> (F) The range specified in a character class had a minimum character greater than the maximum character. One possibility is that you forgot the { } from your ending \x{} - \x without the curly braces can go only up to ff. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Invalid range "%s" in transliteration operator

> (F) The range specified in the tr/// or y/// operator had a minimum character greater than the maximum character. See *perlop*.

Invalid separator character %s in attribute list

> (F) Something other than a colon or whitespace was seen between the elements of an attribute list. If the previous attribute had a parenthesised parameter list, perhaps that list was terminated too soon. See *attributes*.

Invalid separator character %s in PerlIO layer specification %s

> (W layer) When pushing layers onto the Perl I/O system, something other than a colon or whitespace was seen between the elements of a layer list. If the previous attribute had a parenthesised parameter list, perhaps that list was terminated too soon.

Invalid type '%s' in %s

> (F) The given character is not a valid pack or unpack type. See *"pack" in perlfunc*. (W) The given character is not a valid pack or unpack type but used to be silently ignored.

Invalid version format (multiple underscores)

> (F) Versions may contain at most a single underscore, which signals that the version is a beta release. See *version* for the allowed version formats.

Invalid version format (underscores before decimal)

> (F) Versions may not contain decimals after the optional underscore. See *version* for the allowed version formats.

ioctl is not implemented

> (F) Your machine apparently doesn't implement ioctl(), which is pretty strange for a machine that supports C.

ioctl() on unopened %s

> (W unopened) You tried ioctl() on a filehandle that was never opened. Check you control flow and number of arguments.

IO layers (like "%s") unavailable

> (F) Your Perl has not been configured to have PerlIO, and therefore you cannot use IO layers. To have PerlIO Perl must be configured with 'useperlio'.

IO::Socket::atmark not implemented on this architecture

> (F) Your machine doesn't implement the sockatmark() functionality, neither as a system call or an ioctl call (SIOCATMARK).

$* is no longer supported

> (S deprecated, syntax) The special variable $*, deprecated in older perls, has been removed as of 5.9.0 and is no longer supported. In previous versions of perl the use of $* enabled or disabled multi-line matching within a string.

> Instead of using $* you should use the /m (and maybe /s) regexp modifiers. (In older versions: when $* was set to a true value then all regular expressions behaved as if they were written using /m.)

**$# is no longer supported**

(S deprecated, syntax) The special variable $#, deprecated in older perls, has been removed as of 5.9.3 and is no longer supported. You should use the printf/sprintf functions instead.

**`%s' is not a code reference**

(W overload) The second (fourth, sixth, ...) argument of overload::constant needs to be a code reference. Either an anonymous subroutine, or a reference to a subroutine.

**`%s' is not an overloadable type**

(W overload) You tried to overload a constant type the overload package is unaware of.

**junk on end of regexp**

(P) The regular expression parser is confused.

**Label not found for "last %s"**

(F) You named a loop to break out of, but you're not currently in a loop of that name, not even if you count where you were called from. See *"last" in perlfunc*.

**Label not found for "next %s"**

(F) You named a loop to continue, but you're not currently in a loop of that name, not even if you count where you were called from. See *"last" in perlfunc*.

**Label not found for "redo %s"**

(F) You named a loop to restart, but you're not currently in a loop of that name, not even if you count where you were called from. See *"last" in perlfunc*.

**leaving effective %s failed**

(F) While under the `use filetest` pragma, switching the real and effective uids or gids failed.

**length/code after end of string in unpack**

(F) While unpacking, the string buffer was already used up when an unpack length/code combination tried to obtain more data. This results in an undefined value for the length. See *"pack" in perlfunc*.

**listen() on closed socket %s**

(W closed) You tried to do a listen on a closed socket. Did you forget to check the return value of your socket() call? See *"listen" in perlfunc*.

**Lookbehind longer than %d not implemented in regex m/%s/**

(F) There is currently a limit on the length of string which lookbehind can handle. This restriction may be eased in a future release.

**lstat() on filehandle %s**

(W io) You tried to do an lstat on a filehandle. What did you mean by that? lstat() makes sense only on filenames. (Perl did a fstat() instead on the filehandle.)

**Lvalue subs returning %s not implemented yet**

(F) Due to limitations in the current implementation, array and hash values cannot be returned in subroutines used in lvalue context. See *"Lvalue subroutines" in perlsub*.

**Malformed integer in [] in pack**

(F) Between the brackets enclosing a numeric repeat count only digits are permitted. See *"pack" in perlfunc*.

**Malformed integer in [] in unpack**

(F) Between the brackets enclosing a numeric repeat count only digits are permitted. See *"pack" in perlfunc*.

Malformed PERLLIB_PREFIX

(F) An error peculiar to OS/2. PERLLIB_PREFIX should be of the form

```
prefix1;prefix2
```

or prefix1 prefix2

with nonempty prefix1 and prefix2. If `prefix1` is indeed a prefix of a builtin library search path, prefix2 is substituted. The error may appear if components are not found, or are too long. See "PERLLIB_PREFIX" in *perlos2*.

Malformed prototype for %s: %s

(F) You tried to use a function with a malformed prototype. The syntax of function prototypes is given a brief compile-time check for obvious errors like invalid characters. A more rigorous check is run when the function is called.

Malformed UTF-8 character (%s)

(S utf8) (F) Perl detected a string that didn't comply with UTF-8 encoding rules, even though it had the UTF8 flag on.

One possible cause is that you set the UTF8 flag yourself for data that you thought to be in UTF-8 but it wasn't (it was for example legacy 8-bit data). To guard against this, you can use Encode::decode_utf8.

If you use the `:encoding(UTF-8)` PerlIO layer for input, invalid byte sequences are handled gracefully, but if you use `:utf8`, the flag is set without validating the data, possibly resulting in this error message.

See also *"Handling Malformed Data" in Encode*.

Malformed UTF-16 surrogate

Perl thought it was reading UTF-16 encoded character data but while doing it Perl met a malformed Unicode surrogate.

Malformed UTF-8 string in pack

(F) You tried to pack something that didn't comply with UTF-8 encoding rules and perl was unable to guess how to make more progress.

Malformed UTF-8 string in unpack

(F) You tried to unpack something that didn't comply with UTF-8 encoding rules and perl was unable to guess how to make more progress.

Malformed UTF-8 string in '%c' format in unpack

(F) You tried to unpack something that didn't comply with UTF-8 encoding rules and perl was unable to guess how to make more progress.

Maximal count of pending signals (%s) exceeded

(F) Perl aborted due to a too important number of signals pending. This usually indicates that your operating system tried to deliver signals too fast (with a very high priority), starving the perl process from resources it would need to reach a point where it can process signals safely. (See *"Deferred Signals (Safe Signals)" in perlipc*.)

%s matches null string many times in regex; marked by <-- HERE in m/%s/

(W regexp) The pattern you've specified would be an infinite loop if the regular expression engine didn't specifically check for that. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

**"%s" may clash with future reserved word**

> (W) This warning may be due to running a perl5 script through a perl4 interpreter, especially if the word that is being warned about is "use" or "my".

**% may not be used in pack**

> (F) You can't pack a string by supplying a checksum, because the checksumming process loses information, and you can't go the other way. See *"unpack" in perlfunc*.

**Method for operation %s not found in package %s during blessing**

> (F) An attempt was made to specify an entry in an overloading table that doesn't resolve to a valid subroutine. See *overload*.

**Method %s not permitted**

> See Server error.

**Might be a runaway multi-line %s string starting on line %d**

> (S) An advisory indicating that the previous error may have been caused by a missing delimiter on a string or pattern, because it eventually ended earlier on the current line.

**Misplaced _ in number**

> (W syntax) An underscore (underbar) in a numeric constant did not separate two digits.

**Missing argument to -%c**

> (F) The argument to the indicated command line switch must follow immediately after the switch, without intervening spaces.

**Missing %sbrace%s on \N{}**

> (F) Wrong syntax of character name literal \N{charname} within double-quotish context.

**Missing comma after first argument to %s function**

> (F) While certain functions allow you to specify a filehandle or an "indirect object" before the argument list, this ain't one of them.

**Missing command in piped open**

> (W pipe) You used the open(FH, "| command") or open(FH, "command |") construction, but the command was missing or blank.

**Missing control char name in \c**

> (F) A double-quoted string ended with "\c", without the required control character name.

**Missing name in "my sub"**

> (F) The reserved syntax for lexically scoped subroutines requires that they have a name with which they can be found.

**Missing $ on loop variable**

> (F) Apparently you've been programming in **csh** too much. Variables are always mentioned with the $ in Perl, unlike in the shells, where it can vary from one line to the next.

**(Missing operator before %s?)**

> (S syntax) This is an educated guess made in conjunction with the message "%s found where operator expected". Often the missing operator is a comma.

**Missing right brace on %s**

> (F) Missing right brace in \x{...}, \p{...} or \P{...}.

**Missing right curly or square bracket**

(F) The lexer counted more opening curly or square brackets than closing ones. As a general rule, you'll find it's missing near the place you were last editing.

(Missing semicolon on previous line?)

(S syntax) This is an educated guess made in conjunction with the message "%s found where operator expected". Don't automatically put a semicolon on the previous line just because you saw this message.

Modification of a read-only value attempted

(F) You tried, directly or indirectly, to change the value of a constant. You didn't, of course, try "2 = 1", because the compiler catches that. But an easy way to do the same thing is:

```
sub mod { $_[0] = 1 }
mod(2);
```

Another way is to assign to a substr() that's off the end of the string.

Yet another way is to assign to a `foreach` loop *VAR* when *VAR* is aliased to a constant in the look *LIST*:

```
$x = 1;
foreach my $n ($x, 2) {
    $n *= 2; # modifies the $x, but fails on attempt to
modify the 2
    }
```

Modification of non-creatable array value attempted, %s

(F) You tried to make an array value spring into existence, and the subscript was probably negative, even counting from end of the array backwards.

Modification of non-creatable hash value attempted, %s

(P) You tried to make a hash value spring into existence, and it couldn't be created for some peculiar reason.

Module name must be constant

(F) Only a bare module name is allowed as the first argument to a "use".

Module name required with -%c option

(F) The `-M` or `-m` options say that Perl should load some module, but you omitted the name of the module. Consult *perlrun* for full details about `-M` and `-m`.

More than one argument to open

(F) The `open` function has been asked to open multiple files. This can happen if you are trying to open a pipe to a command that takes a list of arguments, but have forgotten to specify a piped open mode. See *"open" in perlfunc* for details.

msg%s not implemented

(F) You don't have System V message IPC on your system.

Multidimensional syntax %s not supported

(W syntax) Multidimensional arrays aren't written like $foo[1,2,3]. They're written like $foo[1][2][3], as in C.

'/' must follow a numeric type in unpack

(F) You had an unpack template that contained a '/', but this did not follow some unpack specification producing a numeric value. See *"pack" in perlfunc*.

"my sub" not yet implemented

(F) Lexically scoped subroutines are not yet implemented. Don't try that yet.

"%s" variable %s can't be in a package

(F) Lexically scoped variables aren't in a package, so it doesn't make sense to try to declare one with a package qualifier on the front. Use local() if you want to localize a package variable.

Name "%s::%s" used only once: possible typo

(W once) Typographical errors often show up as unique variable names. If you had a good reason for having a unique name, then just mention it again somehow to suppress the message. The `our` declaration is provided for this purpose.

NOTE: This warning detects symbols that have been used only once so $c, @c, %c, *c, &c, sub c{}, c(), and c (the filehandle or format) are considered the same; if a program uses $c only once but also uses any of the others it will not trigger this warning.

Negative '/' count in unpack

(F) The length count obtained from a length/code unpack operation was negative. See *"pack" in perlfunc*.

Negative length

(F) You tried to do a read/write/send/recv operation with a buffer length that is less than 0. This is difficult to imagine.

Negative offset to vec in lvalue context

(F) When `vec` is called in an lvalue context, the second argument must be greater than or equal to zero.

Nested quantifiers in regex; marked by <-- HERE in m/%s/

(F) You can't quantify a quantifier without intervening parentheses. So things like ** or +* or ?* are illegal. The <-- HERE shows in the regular expression about where the problem was discovered.

Note that the minimal matching quantifiers, `*?`, `+?`, and `??` appear to be nested quantifiers, but aren't. See *perlre*.

%s never introduced

(S internal) The symbol in question was declared but somehow went out of scope before it could possibly have been used.

next::method/next::can/maybe::next::method cannot find enclosing method

(F) `next::method` needs to be called within the context of a real method in a real package, and it could not find such a context. See *mro*.

No %s allowed while running setuid

(F) Certain operations are deemed to be too insecure for a setuid or setgid script to even be allowed to attempt. Generally speaking there will be another way to do what you want that is, if not secure, at least securable. See *perlsec*.

No comma allowed after %s

(F) A list operator that has a filehandle or "indirect object" is not allowed to have a comma between that and the following arguments. Otherwise it'd be just another one of the arguments.

One possible cause for this is that you expected to have imported a constant to your name space with **use** or **import** while no such importing took place, it may for example be that your operating system does not support that particular constant. Hopefully you did use an explicit import list for the constants you expect to see, please see *"use" in perlfunc* and *"import" in*

*perlfunc*. While an explicit import list would probably have caught this error earlier it naturally does not remedy the fact that your operating system still does not support that constant. Maybe you have a typo in the constants of the symbol import list of **use** or **import** or in the constant name at the line where this error was triggered?

No command into which to pipe on command line

> (F) An error peculiar to VMS. Perl handles its own command line redirection, and found a '|' at the end of the command line, so it doesn't know where you want to pipe the output from this command.

No DB::DB routine defined

> (F) The currently executing code was compiled with the **-d** switch, but for some reason the current debugger (e.g. *perl5db.pl* or a `Devel::` module) didn't define a routine to be called at the beginning of each statement.

No dbm on this machine

> (P) This is counted as an internal error, because every machine should supply dbm nowadays, because Perl comes with SDBM. See *SDBM_File*.

No DB::sub routine defined

> (F) The currently executing code was compiled with the **-d** switch, but for some reason the current debugger (e.g. *perl5db.pl* or a `Devel::` module) didn't define a `DB::sub` routine to be called at the beginning of each ordinary subroutine call.

No **-e** allowed in setuid scripts

> (F) A setuid script can't be specified by the user.

No error file after 2> or 2>> on command line

> (F) An error peculiar to VMS. Perl handles its own command line redirection, and found a '2>' or a '2>>' on the command line, but can't find the name of the file to which to write data destined for stderr.

No group ending character '%c' found in template

> (F) A pack or unpack template has an opening '(' or '[' without its matching counterpart. See *"pack" in perlfunc*.

No input file after < on command line

> (F) An error peculiar to VMS. Perl handles its own command line redirection, and found a '<' on the command line, but can't find the name of the file from which to read data for stdin.

No #! line

> (F) The setuid emulator requires that scripts have a well-formed #! line even on machines that don't support the #! construct.

No next::method '%s' found for %s

> (F) `next::method` found no further instances of this method name in the remaining packages of the MRO of this class. If you don't want it throwing an exception, use `maybe::next::method` or `next::can`. See *mro*.

"no" not allowed in expression

> (F) The "no" keyword is recognized and executed at compile time, and returns no useful value. See *perlmod*.

No output file after > on command line

> (F) An error peculiar to VMS. Perl handles its own command line redirection, and found a lone '>' at the end of the command line, so it doesn't know where you wanted to redirect stdout.

No output file after > or >> on command line

> (F) An error peculiar to VMS. Perl handles its own command line redirection, and found a '>' or a '>>' on the command line, but can't find the name of the file to which to write data destined for stdout.

No package name allowed for variable %s in "our"

> (F) Fully qualified variable names are not allowed in "our" declarations, because that doesn't make much sense under existing semantics. Such syntax is reserved for future extensions.

No Perl script found in input

> (F) You called `perl -x`, but no line was found in the file beginning with #! and containing the word "perl".

No setregid available

> (F) Configure didn't find anything resembling the setregid() call for your system.

No setreuid available

> (F) Configure didn't find anything resembling the setreuid() call for your system.

No %s specified for -%c

> (F) The indicated command line switch needs a mandatory argument, but you haven't specified one.

No such class field "%s" in variable %s of type %s

> (F) You tried to access a key from a hash through the indicated typed variable but that key is not allowed by the package of the same type. The indicated package has restricted the set of allowed keys using the *fields* pragma.

No such class %s

> (F) You provided a class qualifier in a "my", "our" or "state" declaration, but this class doesn't exist at this point in your program.

No such hook: %s

> (F) You specified a signal hook that was not recognized by Perl. Currently, Perl accepts `__DIE__` and `__WARN__` as valid signal hooks

No such pipe open

> (P) An error peculiar to VMS. The internal routine my_pclose() tried to close a pipe which hadn't been opened. This should have been caught earlier as an attempt to close an unopened filehandle.

No such signal: SIG%s

> (W signal) You specified a signal name as a subscript to %SIG that was not recognized. Say `kill -l` in your shell to see the valid signal names on your system.

Not a CODE reference

> (F) Perl was trying to evaluate a reference to a code value (that is, a subroutine), but found a reference to something else instead. You can use the ref() function to find out what kind of ref it really was. See also *perlref*.

Not a format reference

> (F) I'm not sure how you managed to generate a reference to an anonymous format, but this indicates you did, and that it didn't exist.

Not a GLOB reference

> (F) Perl was trying to evaluate a reference to a "typeglob" (that is, a symbol table entry that

looks like `*foo`), but found a reference to something else instead. You can use the ref() function to find out what kind of ref it really was. See *perlref*.

Not a HASH reference

> (F) Perl was trying to evaluate a reference to a hash value, but found a reference to something else instead. You can use the ref() function to find out what kind of ref it really was. See *perlref*.

Not an ARRAY reference

> (F) Perl was trying to evaluate a reference to an array value, but found a reference to something else instead. You can use the ref() function to find out what kind of ref it really was. See *perlref*.

Not a perl script

> (F) The setuid emulator requires that scripts have a well-formed #! line even on machines that don't support the #! construct. The line must mention perl.

Not a SCALAR reference

> (F) Perl was trying to evaluate a reference to a scalar value, but found a reference to something else instead. You can use the ref() function to find out what kind of ref it really was. See *perlref*.

Not a subroutine reference

> (F) Perl was trying to evaluate a reference to a code value (that is, a subroutine), but found a reference to something else instead. You can use the ref() function to find out what kind of ref it really was. See also *perlref*.

Not a subroutine reference in overload table

> (F) An attempt was made to specify an entry in an overloading table that doesn't somehow point to a valid subroutine. See *overload*.

Not enough arguments for %s

> (F) The function requires more arguments than you specified.

Not enough format arguments

> (W syntax) A format specified more picture fields than the next line supplied. See *perlform*.

%s: not found

> (A) You've accidentally run your script through the Bourne shell instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

no UTC offset information; assuming local time is UTC

> (S) A warning peculiar to VMS. Perl was unable to find the local timezone offset, so it's assuming that local system time is equivalent to UTC. If it's not, define the logical name *SYS$TIMEZONE_DIFFERENTIAL* to translate to the number of seconds which need to be added to UTC to get local time.

Non-string passed as bitmask

> (W misc) A number has been passed as a bitmask argument to select(). Use the vec() function to construct the file descriptor bitmasks for select. See *"select" in perlfunc*

Null filename used

> (F) You can't require the null filename, especially because on many machines that means the current directory! See *"require" in perlfunc*.

NULL OP IN RUN

(P debugging) Some internal routine called run() with a null opcode pointer.

**Null picture in formline**

(F) The first argument to formline must be a valid format picture specification. It was found to be empty, which probably means you supplied it an uninitialized value. See *perlform*.

**Null realloc**

(P) An attempt was made to realloc NULL.

**NULL regexp argument**

(P) The internal pattern matching routines blew it big time.

**NULL regexp parameter**

(P) The internal pattern matching routines are out of their gourd.

**Number too long**

(F) Perl limits the representation of decimal numbers in programs to about 250 characters. You've exceeded that length. Future versions of Perl are likely to eliminate this arbitrary limitation. In the meantime, try using scientific notation (e.g. "1e6" instead of "1_000_000").

**Octal number in vector unsupported**

(F) Numbers with a leading `0` are not currently allowed in vectors. The octal number interpretation of such numbers may be supported in a future version.

**Octal number > 037777777777 non-portable**

(W portable) The octal number you specified is larger than 2**32-1 (4294967295) and therefore non-portable between systems. See *perlport* for more on portability concerns.

See also *perlport* for writing portable code.

**Odd number of arguments for overload::constant**

(W overload) The call to overload::constant contained an odd number of arguments. The arguments should come in pairs.

**Odd number of elements in anonymous hash**

(W misc) You specified an odd number of elements to initialize a hash, which is odd, because hashes come in key/value pairs.

**Odd number of elements in hash assignment**

(W misc) You specified an odd number of elements to initialize a hash, which is odd, because hashes come in key/value pairs.

**Offset outside string**

(F, W layer) You tried to do a read/write/send/recv/seek operation with an offset pointing outside the buffer. This is difficult to imagine. The sole exceptions to this are that zero padding will take place when going past the end of the string when either `sysread()`ing a file, or when seeking past the end of a scalar opened for I/O (in anticipation of future reads and to imitate the behaviour with real files).

**%s() on unopened %s**

(W unopened) An I/O operation was attempted on a filehandle that was never initialized. You need to do an open(), a sysopen(), or a socket() call, or call a constructor from the FileHandle package.

**-%s on unopened filehandle %s**

(W unopened) You tried to invoke a file test operator on a filehandle that isn't open. Check your control flow. See also *"-X" in perlfunc*.

oops: oopsAV

> (S internal) An internal warning that the grammar is screwed up.

oops: oopsHV

> (S internal) An internal warning that the grammar is screwed up.

Opening dirhandle %s also as a file

> (W io deprecated) You used open() to associate a filehandle to a symbol (glob or scalar) that already holds a dirhandle. Although legal, this idiom might render your code confusing and is deprecated.

Opening filehandle %s also as a directory

> (W io deprecated) You used opendir() to associate a dirhandle to a symbol (glob or scalar) that already holds a filehandle. Although legal, this idiom might render your code confusing and is deprecated.

Operation "%s": no method found, %s

> (F) An attempt was made to perform an overloaded operation for which no handler was defined. While some handlers can be autogenerated in terms of other handlers, there is no default handler for any operation, unless `fallback` overloading key is specified to be true. See *overload*.

Operator or semicolon missing before %s

> (S ambiguous) You used a variable or subroutine call where the parser was expecting an operator. The parser has assumed you really meant to use an operator, but this is highly likely to be incorrect. For example, if you say "*foo *foo" it will be interpreted as if you said "*foo * 'foo'".

"our" variable %s redeclared

> (W misc) You seem to have already declared the same global once before in the current lexical scope.

Out of memory!

> (X) The malloc() function returned 0, indicating there was insufficient remaining memory (or virtual memory) to satisfy the request. Perl has no option but to exit immediately.
>
> At least in Unix you may be able to get past this by increasing your process datasize limits: in csh/tcsh use `limit` and `limit datasize n` (where `n` is the number of kilobytes) to check the current limits and change them, and in ksh/bash/zsh use `ulimit -a` and `ulimit -d n`, respectively.

Out of memory during %s extend

> (X) An attempt was made to extend an array, a list, or a string beyond the largest possible memory allocation.

Out of memory during "large" request for %s

> (F) The malloc() function returned 0, indicating there was insufficient remaining memory (or virtual memory) to satisfy the request. However, the request was judged large enough (compile-time default is 64K), so a possibility to shut down by trapping this error is granted.

Out of memory during request for %s

> (X|F) The malloc() function returned 0, indicating there was insufficient remaining memory (or virtual memory) to satisfy the request.
>
> The request was judged to be small, so the possibility to trap it depends on the way perl was compiled. By default it is not trappable. However, if compiled for this, Perl may use the contents of `$^M` as an emergency pool after die()ing with this message. In this case the error

is trappable *once*, and the error message will include the line and file where the failed request happened.

Out of memory during ridiculously large request

(F) You can't allocate more than 2^31+"small amount" bytes. This error is most likely to be caused by a typo in the Perl program. e.g., `$arr[time]` instead of `$arr[$time]`.

Out of memory for yacc stack

(F) The yacc parser wanted to grow its stack so it could continue parsing, but realloc() wouldn't give it more memory, virtual or otherwise.

'.' outside of string in pack

(F) The argument to a '.' in your template tried to move the working position to before the start of the packed string being built.

'@' outside of string in unpack

(F) You had a template that specified an absolute position outside the string being unpacked. See *"pack" in perlfunc*.

'@' outside of string with malformed UTF-8 in unpack

(F) You had a template that specified an absolute position outside the string being unpacked. The string being unpacked was also invalid UTF-8. See *"pack" in perlfunc*.

%s package attribute may clash with future reserved word: %s

(W reserved) A lowercase attribute name was used that had a package-specific handler. That name might have a meaning to Perl itself some day, even though it doesn't yet. Perhaps you should use a mixed-case attribute name, instead. See *attributes*.

pack/unpack repeat count overflow

(F) You can't specify a repeat count so large that it overflows your signed integers. See *"pack" in perlfunc*.

page overflow

(W io) A single call to write() produced more lines than can fit on a page. See *perlform*.

panic: %s

(P) An internal error.

panic: attempt to call %s in %s

(P) One of the file test operators entered a code branch that calls an ACL related-function, but that function is not available on this platform. Earlier checks mean that it should not be possible to enter this branch on this platform.

panic: ck_grep

(P) Failed an internal consistency check trying to compile a grep.

panic: ck_split

(P) Failed an internal consistency check trying to compile a split.

panic: corrupt saved stack index

(P) The savestack was requested to restore more localized values than there are in the savestack.

panic: del_backref

(P) Failed an internal consistency check while trying to reset a weak reference.

panic: Devel::DProf inconsistent subroutine return

> (P) Devel::DProf called a subroutine that exited using goto(LABEL), last(LABEL) or next(LABEL). Leaving that way a subroutine called from an XSUB will lead very probably to a crash of the interpreter. This is a bug that will hopefully one day get fixed.

panic: die %s

> (P) We popped the context stack to an eval context, and then discovered it wasn't an eval context.

panic: do_subst

> (P) The internal pp_subst() routine was called with invalid operational data.

panic: do_trans_%s

> (P) The internal do_trans routines were called with invalid operational data.

panic: fold_constants JMPENV_PUSH returned %d

> (P) While attempting folding constants an exception other than an `eval` failure was caught.

panic: frexp

> (P) The library function frexp() failed, making printf("%f") impossible.

panic: goto

> (P) We popped the context stack to a context with the specified label, and then discovered it wasn't a context we know how to do a goto in.

panic: hfreeentries failed to free hash

> (P) The internal routine used to clear a hashes entries tried repeatedly, but each time something added more entries to the hash. Most likely the hash contains an object with a reference back to the hash and a destructor that adds a new object to the hash.

panic: INTERPCASEMOD

> (P) The lexer got into a bad state at a case modifier.

panic: INTERPCONCAT

> (P) The lexer got into a bad state parsing a string with brackets.

panic: kid popen errno read

> (F) forked child returned an incomprehensible message about its errno.

panic: last

> (P) We popped the context stack to a block context, and then discovered it wasn't a block context.

panic: leave_scope clearsv

> (P) A writable lexical variable became read-only somehow within the scope.

panic: leave_scope inconsistency

> (P) The savestack probably got out of sync. At least, there was an invalid enum on the top of it.

panic: magic_killbackrefs

> (P) Failed an internal consistency check while trying to reset all weak references to an object.

panic: malloc

> (P) Something requested a negative number of bytes of malloc.

---

panic: memory wrap

(P) Something tried to allocate more memory than possible.

panic: pad_alloc

(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_free curpad

(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_free po

(P) An invalid scratch pad offset was detected internally.

panic: pad_reset curpad

(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_sv po

(P) An invalid scratch pad offset was detected internally.

panic: pad_swipe curpad

(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_swipe po

(P) An invalid scratch pad offset was detected internally.

panic: pp_iter

(P) The foreach iterator got called in a non-loop context frame.

panic: pp_match%s

(P) The internal pp_match() routine was called with invalid operational data.

panic: pp_split

(P) Something terrible went wrong in setting up for the split.

panic: realloc

(P) Something requested a negative number of bytes of realloc.

panic: restartop

(P) Some internal routine requested a goto (or something like it), and didn't supply the destination.

panic: return

(P) We popped the context stack to a subroutine or eval context, and then discovered it wasn't a subroutine or eval context.

panic: scan_num

(P) scan_num() got called on something that wasn't a number.

panic: sv_chop %s

(P) The sv_chop() routine was passed a position that is not within the scalar's string buffer.

panic: sv_insert

(P) The sv_insert() routine was told to remove more string than there was string.

panic: top_env

> (P) The compiler attempted to do a goto, or something weird like that.

panic: unimplemented op %s (#%d) called

> (P) The compiler is screwed up and attempted to use an op that isn't permitted at run time.

panic: utf16_to_utf8: odd bytelen

> (P) Something tried to call utf16_to_utf8 with an odd (as opposed to even) byte length.

panic: yylex

> (P) The lexer got into a bad state while processing a case modifier.

Pattern subroutine nesting without pos change exceeded limit in regex; marked by <-- HERE in m/%s/

> (F) You used a pattern that uses too many nested subpattern calls without consuming any text. Restructure the pattern so text is consumed before the nesting limit is exceeded.
>
> The <-- HERE shows in the regular expression about where the problem was discovered.

Parentheses missing around "%s" list

> (W parenthesis) You said something like
>
> ```
> my $foo, $bar = @_;
> ```
>
> when you meant
>
> ```
> my ($foo, $bar) = @_;
> ```
>
> Remember that "my", "our", "local" and "state" bind tighter than comma.

-p destination: %s

> (F) An error occurred during the implicit output invoked by the -p command-line switch. (This output goes to STDOUT unless you've redirected it with select().)

(perhaps you forgot to load "%s"?)

> (F) This is an educated guess made in conjunction with the message "Can't locate object method \"%s\" via package \"%s\"". It often means that a method requires a package that has not been loaded.

Perl_my_%s() not available

> (F) Your platform has very uncommon byte-order and integer size, so it was not possible to set up some or all fixed-width byte-order conversion functions. This is only a problem when you're using the '<' or '>' modifiers in (un)pack templates. See *"pack" in perlfunc*.

Perl %s required--this is only version %s, stopped

> (F) The module in question uses features of a version of Perl more recent than the currently running version. How long has it been since you upgraded, anyway? See *"require" in perlfunc*.

PERL_SH_DIR too long

> (F) An error peculiar to OS/2. PERL_SH_DIR is the directory to find the sh-shell in. See "PERL_SH_DIR" in *perlos2*.

PERL_SIGNALS illegal: "%s"

> See *"PERL_SIGNALS" in perlrun* for legal values.

perl: warning: Setting locale failed.

> (S) The whole warning message will look something like:
>
> ```
> perl: warning: Setting locale failed.
> ```

---

```
perl: warning: Please check that your locale settings:
        LC_ALL = "En_US",
        LANG = (unset)
    are supported and installed on your system.
 perl: warning: Falling back to the standard locale ("C").
```

Exactly what were the failed locale settings varies. In the above the settings were that the LC_ALL was "En_US" and the LANG had no value. This error means that Perl detected that you and/or your operating system supplier and/or system administrator have set up the so-called locale system but Perl could not use those settings. This was not dead serious, fortunately: there is a "default locale" called "C" that Perl can and will use, the script will be run. Before you really fix the problem, however, you will get the same error message each time you run Perl. How to really fix the problem can be found in *perllocale* section **LOCALE PROBLEMS**.

Permission denied

> (F) The setuid emulator in suidperl decided you were up to no good.

pid %x not a child

> (W exec) A warning peculiar to VMS. Waitpid() was asked to wait for a process which isn't a subprocess of the current process. While this is fine from VMS' perspective, it's probably not what you intended.

'P' must have an explicit size in unpack

> (F) The unpack format P must have an explicit size, not "*".

**-P** not allowed for setuid/setgid script

> (F) The script would have to be opened by the C preprocessor by name, which provides a race condition that breaks security.

POSIX class [:%s:] unknown in regex; marked by <-- HERE in m/%s/

> (F) The class in the character class [: :] syntax is unknown. The <-- HERE shows in the regular expression about where the problem was discovered. Note that the POSIX character classes do **not** have the is prefix the corresponding C interfaces have: in other words, it's [[:print:]], not isprint. See *perlre*.

POSIX getpgrp can't take an argument

> (F) Your system has POSIX getpgrp(), which takes no argument, unlike the BSD version, which takes a pid.

POSIX syntax [%s] belongs inside character classes in regex; marked by <-- HERE in m/%s/

> (W regexp) The character class constructs [: :], [= =], and [. .] go *inside* character classes, the [] are part of the construct, for example: /[012[:alpha:]345]/. Note that [= =] and [. .] are not currently implemented; they are simply placeholders for future extensions and will cause fatal errors. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

POSIX syntax [. .] is reserved for future extensions in regex; marked by <-- HERE in m/%s/

> (F regexp) Within regular expression character classes ([]) the syntax beginning with "[." and ending with ".]" is reserved for future extensions. If you need to represent those character sequences inside a regular expression character class, just quote the square brackets with the backslash: "\[." and ".\]". The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

POSIX syntax [= =] is reserved for future extensions in regex; marked by <-- HERE in m/%s/

> (F) Within regular expression character classes ([]) the syntax beginning with "[=" and ending

with "=]" is reserved for future extensions. If you need to represent those character sequences inside a regular expression character class, just quote the square brackets with the backslash: "\[=" and "=\]". The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Possible attempt to put comments in qw() list

(W qw) qw() lists contain items separated by whitespace; as with literal strings, comment characters are not ignored, but are instead treated as literal data. (You may have used different delimiters than the parentheses shown here; braces are also frequently used.)

You probably wrote something like this:

```
    @list = qw(
 a # a comment
        b # another comment
    );
```

when you should have written this:

```
    @list = qw(
 a
        b
    );
```

If you really want comments, build your list the old-fashioned way, with quotes and commas:

```
    @list = (
        'a',    # a comment
        'b',    # another comment
    );
```

Possible attempt to separate words with commas

(W qw) qw() lists contain items separated by whitespace; therefore commas aren't needed to separate the items. (You may have used different delimiters than the parentheses shown here; braces are also frequently used.)

You probably wrote something like this:

```
    qw! a, b, c !;
```

which puts literal commas into some of the list items. Write it without commas if you don't want them to appear in your data:

```
    qw! a b c !;
```

Possible memory corruption: %s overflowed 3rd argument

(F) An ioctl() or fcntl() returned more than Perl was bargaining for. Perl guesses a reasonable buffer size, but puts a sentinel byte at the end of the buffer just in case. This sentinel byte got clobbered, and Perl assumes that memory is now corrupted. See *"ioctl" in perlfunc*.

Possible precedence problem on bitwise %c operator

(W precedence) Your program uses a bitwise logical operator in conjunction with a numeric comparison operator, like this :

```
    if ($x & $y == 0) { ... }
```

This expression is actually equivalent to $x & ($y == 0), due to the higher precedence of ==. This is probably not what you want. (If you really meant to write this, disable the warning, or, better, put the parentheses explicitly and write $x & ($y == 0)).

---

Possible unintended interpolation of %s in string

> (W ambiguous) You said something like `@foo' in a double-quoted string but there was no array `@foo` in scope at the time. If you wanted a literal @foo, then write it as \@foo; otherwise find out what happened to the array you apparently lost track of.

pragma "attrs" is deprecated, use "sub NAME : ATTRS" instead

> (D deprecated) You have written something like this:

```
sub doit
{
    use attrs qw(locked);
}
```

> You should use the new declaration syntax instead.

```
sub doit : locked
{
    ...
```

> The `use attrs` pragma is now obsolete, and is only provided for backward-compatibility. See *"Subroutine Attributes" in perlsub*.

Precedence problem: open %s should be open(%s)

> (S precedence) The old irregular construct

```
open FOO || die;
```

> is now misinterpreted as

```
open(FOO || die);
```

> because of the strict regularization of Perl 5's grammar into unary and list operators. (The old open was a little of both.) You must put parentheses around the filehandle, or use the new "or" operator instead of "||".

Premature end of script headers

> See Server error.

printf() on closed filehandle %s

> (W closed) The filehandle you're writing to got itself closed sometime before now. Check your control flow.

print() on closed filehandle %s

> (W closed) The filehandle you're printing on got itself closed sometime before now. Check your control flow.

Process terminated by SIG%s

> (W) This is a standard message issued by OS/2 applications, while *nix applications die in silence. It is considered a feature of the OS/2 port. One can easily disable this by appropriate sighandlers, see *"Signals" in perlipc*. See also "Process terminated by SIGTERM/SIGINT" in *perlos2*.

Prototype mismatch: %s vs %s

> (S prototype) The subroutine being declared or defined had previously been declared or defined with a different function prototype.

Prototype not terminated

> (F) You've omitted the closing parenthesis in a function prototype definition.

Quantifier follows nothing in regex; marked by <-- HERE in m/%s/

> (F) You started a regular expression with a quantifier. Backslash it if you meant it literally. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Quantifier in {,} bigger than %d in regex; marked by <-- HERE in m/%s/

> (F) There is currently a limit to the size of the min and max values of the {min,max} construct. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Quantifier unexpected on zero-length expression; marked by <-- HERE in m/%s/

> (W regexp) You applied a regular expression quantifier in a place where it makes no sense, such as on a zero-width assertion. Try putting the quantifier inside the assertion instead. For example, the way to match "abc" provided that it is followed by three repetitions of "xyz" is `/abc(?=(?:xyz){3})/`, not `/abc(?=xyz){3}/`.

> The <-- HERE shows in the regular expression about where the problem was discovered.

Range iterator outside integer range

> (F) One (or both) of the numeric arguments to the range operator ".." are outside the range which can be represented by integers internally. One possible workaround is to force Perl to use magical string increment by prepending "0" to your numbers.

readdir() attempted on invalid dirhandle %s

> (W io) The dirhandle you're reading from is either closed or not really a dirhandle. Check your control flow.

readline() on closed filehandle %s

> (W closed) The filehandle you're reading from got itself closed sometime before now. Check your control flow.

read() on closed filehandle %s

> (W closed) You tried to read from a closed filehandle.

read() on unopened filehandle %s

> (W unopened) You tried to read from a filehandle that was never opened.

Reallocation too large: %lx

> (F) You can't allocate more than 64K on an MS-DOS machine.

realloc() of freed memory ignored

> (S malloc) An internal routine called realloc() on something that had already been freed.

Recompile perl with **-D**DEBUGGING to use **-D** switch

> (F debugging) You can't use the **-D** option unless the code to produce the desired output is compiled into Perl, which entails some overhead, which is why it's currently left out of your copy.

Recursive inheritance detected in package '%s'

> (F) While calculating the method resolution order (MRO) of a package, Perl believes it found an infinite loop in the @ISA hierarchy. This is a crude check that bails out after 100 levels of @ISA depth.

Recursive inheritance detected while looking for method %s

> (F) More than 100 levels of inheritance were encountered while invoking a method. Probably indicates an unintended loop in your inheritance hierarchy.

Reference found where even-sized list expected

> (W misc) You gave a single reference where Perl was expecting a list with an even number of elements (for assignment to a hash). This usually means that you used the anon hash constructor when you meant to use parens. In any case, a hash requires key/value **pairs**.

```
%hash = { one => 1, two => 2, }; # WRONG
%hash = [ qw/ an anon array / ]; # WRONG
%hash = ( one => 1, two => 2, ); # right
%hash = qw( one 1 two 2 );   # also fine
```

Reference is already weak

> (W misc) You have attempted to weaken a reference that is already weak. Doing so has no effect.

Reference miscount in sv_replace()

> (W internal) The internal sv_replace() function was handed a new SV with a reference count of other than 1.

Reference to invalid group 0

> (F) You used \g0 or similar in a regular expression. You may refer to capturing parentheses only with strictly positive integers (normal backreferences) or with strictly negative integers (relative backreferences), but using 0 does not make sense.

Reference to nonexistent group in regex; marked by <-- HERE in m/%s/

> (F) You used something like \7 in your regular expression, but there are not at least seven sets of capturing parentheses in the expression. If you wanted to have the character with value 7 inserted into the regular expression, prepend a zero to make the number at least two digits: \07

> The <-- HERE shows in the regular expression about where the problem was discovered.

Reference to nonexistent or unclosed group in regex; marked by <-- HERE in m/%s/

> (F) You used something like \g{-7} in your regular expression, but there are not at least seven sets of closed capturing parentheses in the expression before where the \g{-7} was located.

> The <-- HERE shows in the regular expression about where the problem was discovered.

Reference to nonexistent named group in regex; marked by <-- HERE in m/%s/

> (F) You used something like \k'NAME' or \k<NAME> in your regular expression, but there is no corresponding named capturing parentheses such as (?'NAME'...) or (?<NAME...). Check if the name has been spelled correctly both in the backreference and the declaration.

> The <-- HERE shows in the regular expression about where the problem was discovered.

(?(DEFINE)....) does not allow branches in regex; marked by <-- HERE in m/%s/

> (F) You used something like (?(DEFINE)...|..) which is illegal. The most likely cause of this error is that you left out a parenthesis inside of the .... part.

> The <-- HERE shows in the regular expression about where the problem was discovered.

regexp memory corruption

> (P) The regular expression engine got confused by what the regular expression compiler gave it.

Regexp out of space

> (P) A "can't happen" error, because safemalloc() should have caught it earlier.

Repeated format line will never terminate (~~ and @# incompatible)

(F) Your format contains the ~~ repeat-until-blank sequence and a numeric field that will never go blank so that the repetition never terminates. You might use ^# instead. See *perlform*.

Reversed %s= operator

(W syntax) You wrote your assignment operator backwards. The = must always comes last, to avoid ambiguity with subsequent unary operators.

rewinddir() attempted on invalid dirhandle %s

(W io) The dirhandle you tried to do a rewinddir() on is either closed or not really a dirhandle. Check your control flow.

Runaway format

(F) Your format contained the ~~ repeat-until-blank sequence, but it produced 200 lines at once, and the 200th line looked exactly like the 199th line. Apparently you didn't arrange for the arguments to exhaust themselves, either by using ^ instead of @ (for scalar variables), or by shifting or popping (for array variables). See *perlform*.

Scalars leaked: %d

(P) Something went wrong in Perl's internal bookkeeping of scalars: not all scalar variables were deallocated by the time Perl exited. What this usually indicates is a memory leak, which is of course bad, especially if the Perl program is intended to be long-running.

Scalar value @%s[%s] better written as $%s[%s]

(W syntax) You've used an array slice (indicated by @) to select a single element of an array. Generally it's better to ask for a scalar value (indicated by $). The difference is that `$foo[&bar]` always behaves like a scalar, both when assigning to it and when evaluating its argument, while `@foo[&bar]` behaves like a list when you assign to it, and provides a list context to its subscript, which can do weird things if you're expecting only one subscript.

On the other hand, if you were actually hoping to treat the array element as a list, you need to look into how references work, because Perl will not magically convert between scalars and lists for you. See *perlref*.

Scalar value @%s{%s} better written as $%s{%s}

(W syntax) You've used a hash slice (indicated by @) to select a single element of a hash. Generally it's better to ask for a scalar value (indicated by $). The difference is that `$foo{&bar}` always behaves like a scalar, both when assigning to it and when evaluating its argument, while `@foo{&bar}` behaves like a list when you assign to it, and provides a list context to its subscript, which can do weird things if you're expecting only one subscript.

On the other hand, if you were actually hoping to treat the hash element as a list, you need to look into how references work, because Perl will not magically convert between scalars and lists for you. See *perlref*.

Script is not setuid/setgid in suidperl

(F) Oddly, the suidperl program was invoked on a script without a setuid or setgid bit set. This doesn't make much sense.

Search pattern not terminated

(F) The lexer couldn't find the final delimiter of a // or m{} construct. Remember that bracketing delimiters count nesting level. Missing the leading $ from a variable $m may cause this error.

Note that since Perl 5.9.0 a // can also be the *defined-or* construct, not just the empty search pattern. Therefore code written in Perl 5.9.0 or later that uses the // as the *defined-or* can be misparsed by pre-5.9.0 Perls as a non-terminated search pattern.

Search pattern not terminated or ternary operator parsed as search pattern

(F) The lexer couldn't find the final delimiter of a ?PATTERN? construct.

The question mark is also used as part of the ternary operator (as in `foo ? 0 : 1`) leading to some ambiguous constructions being wrongly parsed. One way to disambiguate the parsing is to put parentheses around the conditional expression, i.e. `(foo) ? 0 : 1`.

%sseek() on unopened filehandle

(W unopened) You tried to use the seek() or sysseek() function on a filehandle that was either never opened or has since been closed.

seekdir() attempted on invalid dirhandle %s

(W io) The dirhandle you are doing a seekdir() on is either closed or not really a dirhandle. Check your control flow.

select not implemented

(F) This machine doesn't implement the select() system call.

Self-ties of arrays and hashes are not supported

(F) Self-ties are of arrays and hashes are not supported in the current implementation.

Semicolon seems to be missing

(W semicolon) A nearby syntax error was probably caused by a missing semicolon, or possibly some other missing operator, such as a comma.

semi-panic: attempt to dup freed string

(S internal) The internal newSVsv() routine was called to duplicate a scalar that had previously been marked as free.

sem%s not implemented

(F) You don't have System V semaphore IPC on your system.

send() on closed socket %s

(W closed) The socket you're sending to got itself closed sometime before now. Check your control flow.

Sequence (? incomplete in regex; marked by <-- HERE in m/%s/

(F) A regular expression ended with an incomplete extension (?. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Sequence (?%s...) not implemented in regex; marked by <-- HERE in m/%s/

(F) A proposed regular expression extension has the character reserved but has not yet been written. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Sequence (?%s...) not recognized in regex; marked by <-- HERE in m/%s/

(F) You used a regular expression extension that doesn't make sense. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Sequence \\%s... not terminated in regex; marked by <-- HERE in m/%s/

(F) The regular expression expects a mandatory argument following the escape sequence and this has been omitted or incorrectly written.

Sequence (?#... not terminated in regex; marked by <-- HERE in m/%s/

(F) A regular expression comment must be terminated by a closing parenthesis. Embedded parentheses aren't allowed. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Sequence (?{...}) not terminated or not {}-balanced in regex; marked by <-- HERE in m/%s/

(F) If the contents of a (?{...}) clause contains braces, they must balance for Perl to properly detect the end of the clause. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

500 Server error

> See Server error.

Server error

> This is the error message generally seen in a browser window when trying to run a CGI program (including SSI) over the web. The actual error text varies widely from server to server. The most frequently-seen variants are "500 Server error", "Method (something) not permitted", "Document contains no data", "Premature end of script headers", and "Did not produce a valid header".
>
> **This is a CGI error, not a Perl error**.
>
> You need to make sure your script is executable, is accessible by the user CGI is running the script under (which is probably not the user account you tested it under), does not rely on any environment variables (like PATH) from the user it isn't running under, and isn't in a location where the CGI server can't find it, basically, more or less. Please see the following for more information:
>
> ```
> http://www.perl.org/CGI_MetaFAQ.html
> http://www.htmlhelp.org/faq/cgifaq.html
> http://www.w3.org/Security/Faq/
> ```
>
> You should also look at *perlfaq9*.

setegid() not implemented

> (F) You tried to assign to $), and your operating system doesn't support the setegid() system call (or equivalent), or at least Configure didn't think so.

seteuid() not implemented

> (F) You tried to assign to $>, and your operating system doesn't support the seteuid() system call (or equivalent), or at least Configure didn't think so.

setpgrp can't take arguments

> (F) Your system has the setpgrp() from BSD 4.2, which takes no arguments, unlike POSIX setpgid(), which takes a process ID and process group ID.

setrgid() not implemented

> (F) You tried to assign to $(, and your operating system doesn't support the setrgid() system call (or equivalent), or at least Configure didn't think so.

setruid() not implemented

> (F) You tried to assign to $<, and your operating system doesn't support the setruid() system call (or equivalent), or at least Configure didn't think so.

setsockopt() on closed socket %s

> (W closed) You tried to set a socket option on a closed socket. Did you forget to check the return value of your socket() call? See *"setsockopt" in perlfunc*.

Setuid/gid script is writable by world

> (F) The setuid emulator won't run a script that is writable by the world, because the world might have written on it already.

Setuid script not plain file

> (F) The setuid emulator won't run a script that isn't read from a file, but from a socket, a pipe

or another device.

**shm%s not implemented**

>    (F) You don't have System V shared memory IPC on your system.

**!=~ should be !~**

>    (W syntax) The non-matching operator is !~, not !=~. !=~ will be interpreted as the != (numeric not equal) and ~ (1's complement) operators: probably not what you intended.

**<> should be quotes**

>    (F) You wrote `require <file>` when you should have written `require 'file'`.

**/%s/ should probably be written as "%s"**

>    (W syntax) You have used a pattern where Perl expected to find a string, as in the first argument to `join`. Perl will treat the true or false result of matching the pattern against $_ as the string, which is probably not what you had in mind.

**shutdown() on closed socket %s**

>    (W closed) You tried to do a shutdown on a closed socket. Seems a bit superfluous.

**SIG%s handler "%s" not defined**

>    (W signal) The signal handler named in %SIG doesn't, in fact, exist. Perhaps you put it into the wrong package?

**Smart matching a non-overloaded object breaks encapsulation**

>    (F) You should not use the ~~ operator on an object that does not overload it: Perl refuses to use the object's underlying structure for the smart match.

**sort is now a reserved word**

>    (F) An ancient error message that almost nobody ever runs into anymore. But before sort was a keyword, people sometimes used it as a filehandle.

**Sort subroutine didn't return a numeric value**

>    (F) A sort comparison routine must return a number. You probably blew it by not using <=> or `cmp`, or by not using them correctly. See *"sort" in perlfunc*.

**Sort subroutine didn't return single value**

>    (F) A sort comparison subroutine may not return a list value with more or less than one element. See *"sort" in perlfunc*.

**splice() offset past end of array**

>    (W misc) You attempted to specify an offset that was past the end of the array passed to splice(). Splicing will instead commence at the end of the array, rather than past it. If this isn't what you want, try explicitly pre-extending the array by assigning $#array = $offset. See *"splice" in perlfunc*.

**Split loop**

>    (P) The split was looping infinitely. (Obviously, a split shouldn't iterate more times than there are characters of input, which is what happened.) See *"split" in perlfunc*.

**Statement unlikely to be reached**

>    (W exec) You did an exec() with some statement after it other than a die(). This is almost always an error, because exec() never returns unless there was a failure. You probably wanted to use system() instead, which does return. To suppress this warning, put the exec() in a block by itself.

stat() on unopened filehandle %s

> (W unopened) You tried to use the stat() function on a filehandle that was either never opened or has since been closed.

Stub found while resolving method "%s" overloading "%s"

> (P) Overloading resolution over @ISA tree may be broken by importation stubs. Stubs should never be implicitly created, but explicit calls to `can` may break this.

Subroutine %s redefined

> (W redefine) You redefined a subroutine. To suppress this warning, say
>
> ```
>     {
>  no warnings 'redefine';
>  eval "sub name { ... }";
>     }
> ```

Substitution loop

> (P) The substitution was looping infinitely. (Obviously, a substitution shouldn't iterate more times than there are characters of input, which is what happened.) See the discussion of substitution in *"Regexp Quote-Like Operators" in perlop*.

Substitution pattern not terminated

> (F) The lexer couldn't find the interior delimiter of an s/// or s{}{} construct. Remember that bracketing delimiters count nesting level. Missing the leading $ from variable $s may cause this error.

Substitution replacement not terminated

> (F) The lexer couldn't find the final delimiter of an s/// or s{}{} construct. Remember that bracketing delimiters count nesting level. Missing the leading $ from variable $s may cause this error.

substr outside of string

> (W substr),(F) You tried to reference a substr() that pointed outside of a string. That is, the absolute value of the offset was larger than the length of the string. See *"substr" in perlfunc*. This warning is fatal if substr is used in an lvalue context (as the left hand side of an assignment or as a subroutine argument for example).

suidperl is no longer needed since %s

> (F) Your Perl was compiled with **-D**SETUID_SCRIPTS_ARE_SECURE_NOW, but a version of the setuid emulator somehow got run anyway.

sv_upgrade from type %d down to type %d

> (P) Perl tried to force the upgrade an SV to a type which was actually inferior to its current type.

Switch (?(condition)... contains too many branches in regex; marked by <-- HERE in m/%s/

> (F) A (?(condition)if-clause|else-clause) construct can have at most two branches (the if-clause and the else-clause). If you want one or both to contain alternation, such as using `this|that|other`, enclose it in clustering parentheses:
>
> ```
>     (?(condition)(?:this|that|other)|else-clause)
> ```
>
> The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Switch condition not recognized in regex; marked by <-- HERE in m/%s/

(F) If the argument to the (?(...)if-clause|else-clause) construct is a number, it can be only a number. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

switching effective %s is not implemented

(F) While under the `use filetest` pragma, we cannot switch the real and effective uids or gids.

%s syntax

(F) The final summary message when a `perl -c` succeeds.

syntax error

(F) Probably means you had a syntax error. Common reasons include:

```
A keyword is misspelled.
A semicolon is missing.
A comma is missing.
An opening or closing parenthesis is missing.
An opening or closing brace is missing.
A closing quote is missing.
```

Often there will be another error message associated with the syntax error giving more information. (Sometimes it helps to turn on **-w**.) The error message itself often tells you where it was in the line when it decided to give up. Sometimes the actual error is several tokens before this, because Perl is good at understanding random input. Occasionally the line number may be misleading, and once in a blue moon the only way to figure out what's triggering the error is to call `perl -c` repeatedly, chopping away half the program each time to see if the error went away. Sort of the cybernetic version of 20 questions.

syntax error at line %d: `%s' unexpected

(A) You've accidentally run your script through the Bourne shell instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

syntax error in file %s at line %d, next 2 tokens "%s"

(F) This error is likely to occur if you run a perl5 script through a perl4 interpreter, especially if the next 2 tokens are "use strict" or "my $var" or "our $var".

sysread() on closed filehandle %s

(W closed) You tried to read from a closed filehandle.

sysread() on unopened filehandle %s

(W unopened) You tried to read from a filehandle that was never opened.

System V %s is not implemented on this machine

(F) You tried to do something with a function beginning with "sem", "shm", or "msg" but that System V IPC is not implemented in your machine. In some machines the functionality can exist but be unconfigured. Consult your system support.

syswrite() on closed filehandle %s

(W closed) The filehandle you're writing to got itself closed sometime before now. Check your control flow.

-T and -B not implemented on filehandles

(F) Perl can't peek at the stdio buffer of filehandles when it doesn't know about your kind of stdio. You'll have to use a filename instead.

Target of goto is too deeply nested

(F) You tried to use `goto` to reach a label that was too deeply nested for Perl to reach. Perl is doing you a favor by refusing.

tell() on unopened filehandle

(W unopened) You tried to use the tell() function on a filehandle that was either never opened or has since been closed.

telldir() attempted on invalid dirhandle %s

(W io) The dirhandle you tried to telldir() is either closed or not really a dirhandle. Check your control flow.

That use of $[ is unsupported

(F) Assignment to `$[` is now strictly circumscribed, and interpreted as a compiler directive. You may say only one of

```
$[ = 0;
$[ = 1;
...
local $[ = 0;
local $[ = 1;
...
```

This is to prevent the problem of one module changing the array base out from under another module inadvertently. See *"$[" in perlvar*.

The crypt() function is unimplemented due to excessive paranoia

(F) Configure couldn't find the crypt() function on your machine, probably because your vendor didn't supply it, probably because they think the U.S. Government thinks it's a secret, or at least that they will continue to pretend that it is. And if you quote me on that, I will deny it.

The %s function is unimplemented

The function indicated isn't implemented on this architecture, according to the probings of Configure.

The stat preceding %s wasn't an lstat

(F) It makes no sense to test the current stat buffer for symbolic linkhood if the last stat that wrote to the stat buffer already went past the symlink to get to the real file. Use an actual filename instead.

The 'unique' attribute may only be applied to 'our' variables

(F) This attribute was never supported on `my` or `sub` declarations.

This Perl can't reset CRTL environ elements (%s)

This Perl can't set CRTL environ elements (%s=%s)

(W internal) Warnings peculiar to VMS. You tried to change or delete an element of the CRTL's internal environ array, but your copy of Perl wasn't built with a CRTL that contained the setenv() function. You'll need to rebuild Perl with a CRTL that does, or redefine *PERL_ENV_TABLES* (see *perlvms*) so that the environ array isn't the target of the change to %ENV which produced the warning.

thread failed to start: %s

(W threads)(S) The entry point function of threads->create() failed for some reason.

times not implemented

(F) Your version of the C library apparently doesn't do times(). I suspect you're not running on Unix.

"-T" is on the #! line, it must also be used on the command line

> (X) The #! line (or local equivalent) in a Perl script contains the **-T** option, but Perl was not invoked with **-T** in its command line. This is an error because, by the time Perl discovers a **-T** in a script, it's too late to properly taint everything from the environment. So Perl gives up.

> If the Perl script is being executed as a command using the #! mechanism (or its local equivalent), this error can usually be fixed by editing the #! line so that the **-T** option is a part of Perl's first argument: e.g. change `perl -n -T` to `perl -T -n`.

> If the Perl script is being executed as `perl scriptname`, then the **-T** option must appear on the command line: `perl -T scriptname`.

To%s: illegal mapping '%s'

> (F) You tried to define a customized To-mapping for lc(), lcfirst, uc(), or ucfirst() (or their string-inlined versions), but you specified an illegal mapping. See *"User-Defined Character Properties" in perlunicode.*

Too deeply nested ()-groups

> (F) Your template contains ()-groups with a ridiculously deep nesting level.

Too few args to syscall

> (F) There has to be at least one argument to syscall() to specify the system call to call, silly dilly.

Too late for "-%s" option

> (X) The #! line (or local equivalent) in a Perl script contains the **-M**, **-m** or **-C** option.

> In the case of **-M** and **-m**, this is an error because those options are not intended for use inside scripts. Use the `use` pragma instead.

> The **-C** option only works if it is specified on the command line as well (with the same sequence of letters or numbers following). Either specify this option on the command line, or, if your system supports it, make your script executable and run it directly instead of passing it to perl.

Too late to run %s block

> (W void) A CHECK or INIT block is being defined during run time proper, when the opportunity to run them has already passed. Perhaps you are loading a file with `require` or `do` when you should be using `use` instead. Or perhaps you should put the `require` or `do` inside a BEGIN block.

Too many args to syscall

> (F) Perl supports a maximum of only 14 args to syscall().

Too many arguments for %s

> (F) The function requires fewer arguments than you specified.

Too many )'s

> (A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

Too many ('s

> (A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

Trailing \ in regex m/%s/

> (F) The regular expression ends with an unbackslashed backslash. Backslash it. See *perlre*.

Transliteration pattern not terminated

(F) The lexer couldn't find the interior delimiter of a tr/// or tr[][] or y/// or y[][] construct. Missing the leading $ from variables $tr or $y may cause this error.

Transliteration replacement not terminated

(F) The lexer couldn't find the final delimiter of a tr///, tr[][], y/// or y[][] construct.

'%s' trapped by operation mask

(F) You tried to use an operator from a Safe compartment in which it's disallowed. See *Safe*.

truncate not implemented

(F) Your machine doesn't implement a file truncation mechanism that Configure knows about.

Type of arg %d to %s must be %s (not %s)

(F) This function requires the argument in that position to be of a certain type. Arrays must be @NAME or @{EXPR}. Hashes must be %NAME or %{EXPR}. No implicit dereferencing is allowed--use the {EXPR} forms as an explicit dereference. See *perlref*.

umask not implemented

(F) Your machine doesn't implement the umask function and you tried to use it to restrict permissions for yourself (EXPR & 0700).

Unable to create sub named "%s"

(F) You attempted to create or access a subroutine with an illegal name.

Unbalanced context: %d more PUSHes than POPs

(W internal) The exit code detected an internal inconsistency in how many execution contexts were entered and left.

Unbalanced saves: %d more saves than restores

(W internal) The exit code detected an internal inconsistency in how many values were temporarily localized.

Unbalanced scopes: %d more ENTERs than LEAVEs

(W internal) The exit code detected an internal inconsistency in how many blocks were entered and left.

Unbalanced tmps: %d more allocs than frees

(W internal) The exit code detected an internal inconsistency in how many mortal scalars were allocated and freed.

Undefined format "%s" called

(F) The format indicated doesn't seem to exist. Perhaps it's really in another package? See *perlform*.

Undefined sort subroutine "%s" called

(F) The sort comparison routine specified doesn't seem to exist. Perhaps it's in a different package? See *"sort" in perlfunc*.

Undefined subroutine &%s called

(F) The subroutine indicated hasn't been defined, or if it was, it has since been undefined.

Undefined subroutine called

(F) The anonymous subroutine you're trying to call hasn't been defined, or if it was, it has since been undefined.

Undefined subroutine in sort

(F) The sort comparison routine specified is declared but doesn't seem to have been defined yet. See *"sort" in perlfunc*.

Undefined top format "%s" called

(F) The format indicated doesn't seem to exist. Perhaps it's really in another package? See *perlform*.

Undefined value assigned to typeglob

(W misc) An undefined value was assigned to a typeglob, a la `*foo = undef`. This does nothing. It's possible that you really mean `undef *foo`.

%s: Undefined variable

(A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

unexec of %s into %s failed!

(F) The unexec() routine failed for some reason. See your local FSF representative, who probably put it there in the first place.

Unicode character %s is illegal

(W utf8) Certain Unicode characters have been designated off-limits by the Unicode standard and should not be generated. If you really know what you are doing you can turn off this warning by `no warnings 'utf8';`.

Unknown BYTEORDER

(F) There are no byte-swapping functions for a machine with this byte order.

Unknown open() mode '%s'

(F) The second argument of 3-argument open() is not among the list of valid modes: `<`, `>`, `>>`, `+<`, `+>`, `+>>`, `-|`, `|-`, `<&`, `>&`.

Unknown PerlIO layer "%s"

(W layer) An attempt was made to push an unknown layer onto the Perl I/O system. (Layers take care of transforming data between external and internal representations.) Note that some layers, such as `mmap`, are not supported in all environments. If your program didn't explicitly request the failing operation, it may be the result of the value of the environment variable PERLIO.

Unknown process %x sent message to prime_env_iter: %s

(P) An error peculiar to VMS. Perl was reading values for %ENV before iterating over it, and someone else stuck a message in the stream of data Perl expected. Someone's very confused, or perhaps trying to subvert Perl's population of %ENV for nefarious purposes.

Unknown "re" subpragma '%s' (known ones are: %s)

You tried to use an unknown subpragma of the "re" pragma.

Unknown switch condition (?(%.2s in regex; marked by <-- HERE in m/%s/

(F) The condition part of a (?(condition)if-clause|else-clause) construct is not known. The condition may be lookahead or lookbehind (the condition is true if the lookahead or lookbehind is true), a (?{...}) construct (the condition is true if the code evaluates to a true value), or a number (the condition is true if the set of capturing parentheses named by the number matched).

The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Unknown Unicode option letter '%c'

You specified an unknown Unicode option. See *perlrun* documentation of the -C switch for the list of known options.

Unknown Unicode option value %x

You specified an unknown Unicode option. See *perlrun* documentation of the -C switch for the list of known options.

Unknown warnings category '%s'

(F) An error issued by the warnings pragma. You specified a warnings category that is unknown to perl at this point.

Note that if you want to enable a warnings category registered by a module (e.g. use warnings 'File::Find'), you must have imported this module

Unknown verb pattern '%s' in regex; marked by <-- HERE in m/%s/

(F) You either made a typo or have incorrectly put a * quantifier after an open brace in your pattern. Check the pattern and review *perlre* for details on legal verb patterns.

first.

unmatched [ in regex; marked by <-- HERE in m/%s/

(F) The brackets around a character class must match. If you wish to include a closing bracket in a character class, backslash it or put it first. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

unmatched ( in regex; marked by <-- HERE in m/%s/

(F) Unbackslashed parentheses must always be balanced in regular expressions. If you're a vi user, the % key is valuable for finding the matching parenthesis. The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Unmatched right %s bracket

(F) The lexer counted more closing curly or square brackets than opening ones, so you're probably missing a matching opening bracket. As a general rule, you'll find the missing one (so to speak) near the place you were last editing.

Unquoted string "%s" may clash with future reserved word

(W reserved) You used a bareword that might someday be claimed as a reserved word. It's best to put such a word in quotes, or capitalize it somehow, or insert an underbar into it. You might also declare it as a subroutine.

Unrecognized character %s in column %d

(F) The Perl parser has no idea what to do with the specified character in your Perl script (or eval) at the specified column. Perhaps you tried to run a compressed script, a binary program, or a directory as a Perl program.

Unrecognized escape \\%c in character class passed through in regex; marked by <-- HERE in m/%s/

(W regexp) You used a backslash-character combination which is not recognized by Perl inside character classes. The character was understood literally. The <-- HERE shows in the regular expression about where the escape was discovered.

Unrecognized escape \\%c passed through

(W misc) You used a backslash-character combination which is not recognized by Perl. The character was understood literally.

Unrecognized escape \\%c passed through in regex; marked by <-- HERE in m/%s/

(W regexp) You used a backslash-character combination which is not recognized by Perl. The character was understood literally. The <-- HERE shows in the regular expression about

where the escape was discovered.

Unrecognized signal name "%s"

> (F) You specified a signal name to the kill() function that was not recognized. Say `kill -l` in your shell to see the valid signal names on your system.

Unrecognized switch: -%s (-h will show valid options)

> (F) You specified an illegal option to Perl. Don't do that. (If you think you didn't do that, check the #! line to see if it's supplying the bad switch on your behalf.)

Unsuccessful %s on filename containing newline

> (W newline) A file operation was attempted on a filename, and that operation failed, PROBABLY because the filename contained a newline, PROBABLY because you forgot to chomp() it off. See *"chomp" in perlfunc*.

Unsupported directory function "%s" called

> (F) Your machine doesn't support opendir() and readdir().

Unsupported function %s

> (F) This machine doesn't implement the indicated function, apparently. At least, Configure doesn't think so.

Unsupported function fork

> (F) Your version of executable does not support forking.

> Note that under some systems, like OS/2, there may be different flavors of Perl executables, some of which may support fork, some not. Try changing the name you call Perl by to `perl_`, `perl__`, and so on.

Unsupported script encoding %s

> (F) Your program file begins with a Unicode Byte Order Mark (BOM) which declares it to be in a Unicode encoding that Perl cannot read.

Unsupported socket function "%s" called

> (F) Your machine doesn't support the Berkeley socket mechanism, or at least that's what Configure thought.

Unterminated attribute list

> (F) The lexer found something other than a simple identifier at the start of an attribute, and it wasn't a semicolon or the start of a block. Perhaps you terminated the parameter list of the previous attribute too soon. See *attributes*.

Unterminated attribute parameter in attribute list

> (F) The lexer saw an opening (left) parenthesis character while parsing an attribute list, but the matching closing (right) parenthesis character was not found. You may need to add (or remove) a backslash character to get your parentheses to balance. See *attributes*.

Unterminated compressed integer

> (F) An argument to unpack("w",...) was incompatible with the BER compressed integer format and could not be converted to an integer. See *"pack" in perlfunc*.

Unterminated verb pattern in regex; marked by <-- HERE in m/%s/

> (F) You used a pattern of the form `(*VERB)` but did not terminate the pattern with a `)`. Fix the pattern and retry.

Unterminated verb pattern argument in regex; marked by <-- HERE in m/%s/

> (F) You used a pattern of the form `(*VERB:ARG)` but did not terminate the pattern with a `)`.

---

Fix the pattern and retry.

**Unterminated \g{...} pattern in regex; marked by <-- HERE in m/%s/**

(F) You missed a close brace on a \g{..} pattern (group reference) in a regular expression. Fix the pattern and retry.

**Unterminated <> operator**

(F) The lexer saw a left angle bracket in a place where it was expecting a term, so it's looking for the corresponding right angle bracket, and not finding it. Chances are you left some needed parentheses out earlier in the line, and you really meant a "less than".

**untie attempted while %d inner references still exist**

(W untie) A copy of the object returned from `tie` (or `tied`) was still valid when `untie` was called.

**Usage: POSIX::%s(%s)**

(F) You called a POSIX function with incorrect arguments. See *"FUNCTIONS" in POSIX* for more information.

**Usage: Win32::%s(%s)**

(F) You called a Win32 function with incorrect arguments. See *Win32* for more information.

**Useless (?-%s) - don't use /%s modifier in regex; marked by <-- HERE in m/%s/**

(W regexp) You have used an internal modifier such as (?-o) that has no meaning unless removed from the entire regexp:

```
if ($string =~ /(?-o)$pattern/o) { ... }
```

must be written as

```
if ($string =~ /$pattern/) { ... }
```

The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

**Useless localization of %s**

(W syntax) The localization of lvalues such as `local($x=10)` is legal, but in fact the local() currently has no effect. This may change at some point in the future, but in the meantime such code is discouraged.

**Useless (?%s) - use /%s modifier in regex; marked by <-- HERE in m/%s/**

(W regexp) You have used an internal modifier such as (?o) that has no meaning unless applied to the entire regexp:

```
if ($string =~ /(?o)$pattern/) { ... }
```

must be written as

```
if ($string =~ /$pattern/o) { ... }
```

The <-- HERE shows in the regular expression about where the problem was discovered. See *perlre*.

**Useless use of %s in void context**

(W void) You did something without a side effect in a context that does nothing with the return value, such as a statement that doesn't return a value from a block, or the left side of a scalar comma operator. Very often this points not to stupidity on your part, but a failure of Perl to parse your program the way you thought it would. For example, you'd get this if you mixed up your C precedence with Python precedence and said

```
    $one, $two = 1, 2;
```

when you meant to say

```
    ($one, $two) = (1, 2);
```

Another common error is to use ordinary parentheses to construct a list reference when you should be using square or curly brackets, for example, if you say

```
    $array = (1,2);
```

when you should have said

```
    $array = [1,2];
```

The square brackets explicitly turn a list value into a scalar value, while parentheses do not. So when a parenthesized list is evaluated in a scalar context, the comma is treated like C's comma operator, which throws away the left argument, which is not what you want. See *perlref* for more on this.

This warning will not be issued for numerical constants equal to 0 or 1 since they are often used in statements like

```
    1 while sub_with_side_effects();
```

String constants that would normally evaluate to 0 or 1 are warned about.

Useless use of "re" pragma

> (W) You did `use re;` without any arguments. That isn't very useful.

Useless use of sort in scalar context

> (W void) You used sort in scalar context, as in :

```
    my $x = sort @y;
```

> This is not very useful, and perl currently optimizes this away.

Useless use of %s with no values

> (W syntax) You used the push() or unshift() function with no arguments apart from the array, like `push(@x)` or `unshift(@foo)`. That won't usually have any effect on the array, so is completely useless. It's possible in principle that push(@tied_array) could have some effect if the array is tied to a class which implements a PUSH method. If so, you can write it as `push(@tied_array,())` to avoid this warning.

"use" not allowed in expression

> (F) The "use" keyword is recognized and executed at compile time, and returns no useful value. See *perlmod*.

Use of bare << to mean <<"" is deprecated

> (D deprecated, W syntax) You are now encouraged to use the explicitly quoted form if you wish to use an empty line as the terminator of the here-document.

Use of comma-less variable list is deprecated

> (D deprecated, W syntax) The values you give to a format should be separated by commas, not just aligned on a line.

Use of chdir('') or chdir(undef) as chdir() deprecated

> (D deprecated) chdir() with no arguments is documented to change to $ENV{HOME} or $ENV{LOGDIR}. chdir(undef) and chdir('') share this behavior, but that has been deprecated. In future versions they will simply fail.

Be careful to check that what you pass to chdir() is defined and not blank, else you might find yourself in your home directory.

**Use of /c modifier is meaningless in s///**

(W regexp) You used the /c modifier in a substitution. The /c modifier is not presently meaningful in substitutions.

**Use of /c modifier is meaningless without /g**

(W regexp) You used the /c modifier with a regex operand, but didn't use the /g modifier. Currently, /c is meaningful only when /g is used. (This may change in the future.)

**Use of freed value in iteration**

(F) Perhaps you modified the iterated array within the loop? This error is typically caused by code like the following:

```
@a = (3,4);
@a = () for (1,2,@a);
```

You are not supposed to modify arrays while they are being iterated over. For speed and efficiency reasons, Perl internally does not do full reference-counting of iterated items, hence deleting such an item in the middle of an iteration causes Perl to see a freed value.

**Use of *glob{FILEHANDLE} is deprecated**

(D deprecated) You are now encouraged to use the shorter *glob{IO} form to access the filehandle slot within a typeglob.

**Use of /g modifier is meaningless in split**

(W regexp) You used the /g modifier on the pattern for a `split` operator. Since `split` always tries to match the pattern repeatedly, the `/g` has no effect.

**Use of implicit split to @_ is deprecated**

(D deprecated, W syntax) It makes a lot of work for the compiler when you clobber a subroutine's argument list, so it's better if you assign the results of a split() explicitly to an array (or list).

**Use of inherited AUTOLOAD for non-method %s() is deprecated**

(D deprecated) As an (ahem) accidental feature, `AUTOLOAD` subroutines are looked up as methods (using the `@ISA` hierarchy) even when the subroutines to be autoloaded were called as plain functions (e.g. `Foo::bar()`), not as methods (e.g. `Foo->bar()` or `$obj->bar()`).

This bug will be rectified in future by using method lookup only for methods' `AUTOLOAD`s. However, there is a significant base of existing code that may be using the old behavior. So, as an interim step, Perl currently issues an optional warning when non-methods use inherited `AUTOLOAD`s.

The simple rule is: Inheritance will not work when autoloading non-methods. The simple fix for old code is: In any module that used to depend on inheriting `AUTOLOAD` for non-methods from a base class named `BaseClass`, execute `*AUTOLOAD = \&BaseClass::AUTOLOAD` during startup.

In code that currently says `use AutoLoader; @ISA = qw(AutoLoader);` you should remove AutoLoader from @ISA and change `use AutoLoader;` to `use AutoLoader 'AUTOLOAD';`.

**Use of %s in printf format not supported**

(F) You attempted to use a feature of printf that is accessible from only C. This usually means there's a better way to do it in Perl.

**Use of %s is deprecated**

(D deprecated) The construct indicated is no longer recommended for use, generally because there's a better way to do it, and also because the old way has bad side effects.

**Use of -l on filehandle %s**

(W io) A filehandle represents an opened file, and when you opened the file it already went past any symlink you are presumably trying to look for. The operation returned `undef`. Use a filename instead.

**Use of "package" with no arguments is deprecated**

(D deprecated) You used the `package` keyword without specifying a package name. So no namespace is current at all. Using this can cause many otherwise reasonable constructs to fail in baffling ways. `use strict;` instead.

**Use of reference "%s" as array index**

(W misc) You tried to use a reference as an array index; this probably isn't what you mean, because references in numerical context tend to be huge numbers, and so usually indicates programmer error.

If you really do mean it, explicitly numify your reference, like so: `$array[0+$ref]`. This warning is not given for overloaded objects, either, because you can overload the numification and stringification operators and then you assumably know what you are doing.

**Use of reserved word "%s" is deprecated**

(D deprecated) The indicated bareword is a reserved word. Future versions of perl may use it as a keyword, so you're better off either explicitly quoting the word in a manner appropriate for its context of use, or using a different name altogether. The warning can be suppressed for subroutine names by either adding a `&` prefix, or using a package qualifier, e.g. `&our()`, or `Foo::our()`.

**Use of tainted arguments in %s is deprecated**

(W taint, deprecated) You have supplied `system()` or `exec()` with multiple arguments and at least one of them is tainted. This used to be allowed but will become a fatal error in a future version of perl. Untaint your arguments. See *perlsec*.

**Use of uninitialized value%s**

(W uninitialized) An undefined value was used as if it were already defined. It was interpreted as a "" or a 0, but maybe it was a mistake. To suppress this warning assign a defined value to your variables.

To help you figure out what was undefined, perl will try to tell you the name of the variable (if any) that was undefined. In some cases it cannot do this, so it also tells you what operation you used the undefined value in. Note, however, that perl optimizes your program and the operation displayed in the warning may not necessarily appear literally in your program. For example, `"that $foo"` is usually optimized into `"that " . $foo`, and the warning will refer to the `concatenation (.)` operator, even though there is no `.` in your program.

**Using a hash as a reference is deprecated**

(D deprecated) You tried to use a hash as a reference, as in `%foo->{"bar"}` or `%$ref->{"hello"}`. Versions of perl <= 5.6.1 used to allow this syntax, but shouldn't have. It is now deprecated, and will be removed in a future version.

**Using an array as a reference is deprecated**

(D deprecated) You tried to use an array as a reference, as in `@foo->[23]` or `@$ref->[99]`. Versions of perl <= 5.6.1 used to allow this syntax, but shouldn't have. It is now deprecated, and will be removed in a future version.

**UTF-16 surrogate %s**

---

(W utf8) You tried to generate half of an UTF-16 surrogate by requesting a Unicode character between the code points 0xD800 and 0xDFFF (inclusive). That range is reserved exclusively for the use of UTF-16 encoding (by having two 16-bit UCS-2 characters); but Perl encodes its characters in UTF-8, so what you got is a very illegal character. If you really know what you are doing you can turn off this warning by `no warnings 'utf8';`.

Value of %s can be "0"; test with defined()

(W misc) In a conditional expression, you used <HANDLE>, <*> (glob), `each()`, or `readdir()` as a boolean value. Each of these constructs can return a value of "0"; that would make the conditional expression false, which is probably not what you intended. When using these constructs in conditional expressions, test their values with the `defined` operator.

Value of CLI symbol "%s" too long

(W misc) A warning peculiar to VMS. Perl tried to read the value of an %ENV element from a CLI symbol table, and found a resultant string longer than 1024 characters. The return value has been truncated to 1024 characters.

Variable "%s" is not available

(W closure) During compilation, an inner named subroutine or eval is attempting to capture an outer lexical that is not currently available. This can happen for one of two reasons. First, the outer lexical may be declared in an outer anonymous subroutine that has not yet been created. (Remember that named subs are created at compile time, while anonymous subs are created at run-time.) For example,

```
sub { my $a; sub f { $a } }
```

At the time that f is created, it can't capture the current value of $a, since the anonymous subroutine hasn't been created yet. Conversely, the following won't give a warning since the anonymous subroutine has by now been created and is live:

```
sub { my $a; eval 'sub f { $a }' }->();
```

The second situation is caused by an eval accessing a variable that has gone out of scope, for example,

```
    sub f {
 my $a;
 sub { eval '$a' }
    }
    f()->();
```

Here, when the '$a' in the eval is being compiled, f() is not currently being executed, so its $a is not available for capture.

Variable "%s" is not imported%s

(F) While "use strict" in effect, you referred to a global variable that you apparently thought was imported from another module, because something else of the same name (usually a subroutine) is exported by that module. It usually means you put the wrong funny character on the front of your variable.

Variable length lookbehind not implemented in m/%s/

(F) Lookbehind is allowed only for subexpressions whose length is fixed and known at compile time. See *perlre*.

"%s" variable %s masks earlier declaration in same %s

(W misc) A "my", "our" or "state" variable has been redeclared in the current scope or statement, effectively eliminating all access to the previous instance. This is almost always a typographical error. Note that the earlier variable will still exist until the end of the scope or

until all closure referents to it are destroyed.

Variable syntax

(A) You've accidentally run your script through **csh** instead of Perl. Check the #! line, or manually feed your script into Perl yourself.

Variable "%s" will not stay shared

(W closure) An inner (nested) *named* subroutine is referencing a lexical variable defined in an outer named subroutine.

When the inner subroutine is called, it will see the value of the outer subroutine's variable as it was before and during the *first* call to the outer subroutine; in this case, after the first call to the outer subroutine is complete, the inner and outer subroutines will no longer share a common value for the variable. In other words, the variable will no longer be shared.

This problem can usually be solved by making the inner subroutine anonymous, using the `sub {}` syntax. When inner anonymous subs that reference variables in outer subroutines are created, they are automatically rebound to the current values of such variables.

Verb pattern '%s' has a mandatory argument in regex; marked by <-- HERE in m/%s/

(F) You used a verb pattern that requires an argument. Supply an argument or check that you are using the right verb.

Verb pattern '%s' may not have an argument in regex; marked by <-- HERE in m/%s/

(F) You used a verb pattern that is not allowed an argument. Remove the argument or check that you are using the right verb.

Version number must be a constant number

(P) The attempt to translate a `use Module n.n LIST` statement into its equivalent `BEGIN` block found an internal inconsistency with the version number.

Version string '%s' contains invalid data; ignoring: '%s'

(W misc) The version string contains invalid characters at the end, which are being ignored.

Warning: something's wrong

(W) You passed warn() an empty string (the equivalent of `warn ""`) or you called it with no args and `$@` was empty.

Warning: unable to close filehandle %s properly

(S) The implicit close() done by an open() got an error indication on the close(). This usually indicates your file system ran out of disk space.

Warning: Use of "%s" without parentheses is ambiguous

(S ambiguous) You wrote a unary operator followed by something that looks like a binary operator that could also have been interpreted as a term or unary operator. For instance, if you know that the rand function has a default argument of 1.0, and you write

```
rand + 5;
```

you may THINK you wrote the same thing as

```
rand() + 5;
```

but in actual fact, you got

```
rand(+5);
```

So put in parentheses to say what you really mean.

Wide character in %s

(S utf8) Perl met a wide character (>255) when it wasn't expecting one. This warning is by default on for I/O (like print). The easiest way to quiet this warning is simply to add the `:utf8` layer to the output, e.g. `binmode STDOUT, ':utf8'`. Another way to turn off the warning is to add `no warnings 'utf8';` but that is often closer to cheating. In general, you are supposed to explicitly mark the filehandle with an encoding, see *open* and *"binmode" in perlfunc*.

Within []-length '%c' not allowed

(F) The count in the (un)pack template may be replaced by [TEMPLATE] only if TEMPLATE always matches the same amount of packed bytes that can be determined from the template alone. This is not possible if it contains an of the codes @, /, U, u, w or a *-length. Redesign the template.

write() on closed filehandle %s

(W closed) The filehandle you're writing to got itself closed sometime before now. Check your control flow.

%s "\x%s" does not map to Unicode

When reading in different encodings Perl tries to map everything into Unicode characters. The bytes you read in are not legal in this encoding, for example

```
utf8 "\xE4" does not map to Unicode
```

if you try to read in the a-diaereses Latin-1 as UTF-8.

'X' outside of string

(F) You had a (un)pack template that specified a relative position before the beginning of the string being (un)packed. See *"pack" in perlfunc*.

'x' outside of string in unpack

(F) You had a pack template that specified a relative position after the end of the string being unpacked. See *"pack" in perlfunc*.

YOU HAVEN'T DISABLED SET-ID SCRIPTS IN THE KERNEL YET!

(F) And you probably never will, because you probably don't have the sources to your kernel, and your vendor probably doesn't give a rip about what you want. Your best bet is to put a setuid C wrapper around your script.

You need to quote "%s"

(W syntax) You assigned a bareword as a signal handler name. Unfortunately, you already have a subroutine of that name declared, which means that Perl 5 will try to call the subroutine when the assignment is executed, which is probably not what you want. (If it IS what you want, put an & in front.)

Your random numbers are not that random

(F) When trying to initialise the random seed for hashes, Perl could not get any randomness out of your system. This usually indicates Something Very Wrong.

# SEE ALSO

*warnings*, *perllexwarn*.