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 (enabled by default).
(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. E.g. `(W closed)` means a warning in the `closed` category.

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.

Aliasing via reference is experimental

(S experimental::refaliasing) This warning is emitted if you use a reference constructor on the left-hand side of an assignment to alias one variable to another. Simply suppress the warning if you want to use the feature, but know that in doing so you are taking the risk of using an experimental feature which may change or be removed in a future Perl version:

```
no warnings "experimental::refaliasing";
use feature "refaliasing";
\$x = \$y;
```

Allocation too large: `%x`

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

’%c’ allowed only after types `%s` in `%s`

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

alpha->*numify()* is lossy

(W numeric) An alpha version can not be numified without losing information.

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(1) 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`
(S ambiguous) 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.

Ambiguous use of -%s resolved as -&%s()
(S ambiguous) You wrote something like `-foo`, which might be the string `"-foo"`, or a call to the function `foo`, negated. If you meant the string, just write `"-foo"`. If you meant the function call, write `-foo()`.

Ambiguous use of `%c` resolved as operator `%c`
(S ambiguous) %, &, and `*` are both infix operators (modulus, bitwise and, and multiplication) *and* initial special characters (denoting hashes, subroutines and typeglobs), and you said something like `*foo * foo`that might be interpreted as either of them. W e assumed you meant the infix operator, but please try to make it more clear — in the example given, you might write `*foo * foo()` if you really meant to multiply a glob by the result of calling a function.

Ambiguous use of `%c{%s}` resolved to `%c%s`
(W ambiguous) You wrote something like `@{foo}`, which might be asking for the variable `@foo`, or it might be calling a function named foo, and dereferencing it as an array reference. If you wanted the variable, you can just write `@foo`. If you wanted to call the function, write `@{foo()}` ... or you could just not have a variable and a function with the same name, and save yourself a lot of trouble.

Ambiguous use of `%c{%s[...]}` resolved to `%c%s[...]`
Ambiguous use of `%c{%s{...}}` resolved to `%c%s{...}`
(W ambiguous) You wrote something like `${foo[2]}` (where foo represents the name of a Perl keyword), which might be looking for element number 2 of the array named `@foo`, in which case please write `$foo[2]`, or you might have meant to pass an anonymous arrayref to the function named foo, and then do a scalar deref on the value it returns. If you meant that, write `${foo([2])}`.

In regular expressions, the `${foo[2]}` syntax is sometimes necessary to disambiguate between array subscripts and character classes. `/$length[2345]/`, for instance, will be interpreted as `$length` followed by the character class `[2345]`. If an array subscript is what you want, you can avoid the warning by changing `/${length[2345]}/` to the unsightly `/${\$length[2345]}/`, by renaming your array to something that does not coincide with a built-in keyword, or by simply turning off warnings with `no warnings 'ambiguous';`.

'|' 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(1) and ‘‘map’’ in perlfunc(1) for alternatives.

Arg too short for msgsnd

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

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.

Note that for the `Inf` and `NaN` (infinity and not-a-number) the definition of ‘‘numeric’’ is somewhat unusual: the strings themselves (like ‘‘Inf’’) are considered numeric, and anything following them is considered non-numeric.

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.

Argument ‘‘%s’’ treated as 0 in increment (++)

(W numeric) The indicated string was fed as an argument to the `++` operator which expects either a number or a string matching `/^[a-zA-Z]*[0-9]*\z/`. See ‘‘Auto-increment and Auto-decrement’’ in perlop(1) for details.

Array passed to stat will be coerced to a scalar%s

(W syntax) You called *stat()* on an array, but the array will be coerced to a scalar - the number of elements in the array.

assertion botched: %s

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

Assertion %s failed: file ‘‘%s’’, line %d

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

Assigned value is not a reference

(F) You tried to assign something that was not a reference to an lvalue reference (e.g., `\$x = $y`). If you meant to make `$x` an alias to `$y`, use `\$x = \$y`.

Assigned value is not %s reference

(F) You tried to assign a reference to a reference constructor, but the two references were not of the same type. You cannot alias a scalar to an array, or an array to a hash; the two types must match.

```
\$x = \@y; # error
\@x = \%y; # error
$y = [];
\$x = $y; # error; did you mean \$y?
```

Assigning non-zero to $[is no longer possible
(F) When the ‘‘array_base’’ feature is disabled (e.g., under `use v5.16;`) the special variable `$[`, which is deprecated, is now a fixed zero value.

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.

Assuming NOT a POSIX class since %s in regex; marked by <--HERE in m/%s/
(W regexp) You had something like these:

```
[[:alnum]]
[[:digit:xyz]
```

They look like they might have been meant to be the POSIX classes `[:alnum:]` or `[:digit:]`. If so, they should be written:

```
[[:alnum:]]
[[:digit:]xyz]
```

Since these aren’t legal POSIX class specifications, but are legal bracketed character classes, Perl treats them as the latter. In the first example, it matches the characters `":"`, `"["`, `"a"`, `"l"`, `"m"`, `"n"`, and `"u"`.

If these weren’t meant to be POSIX classes, this warning message is spurious, and can be suppressed by reordering things, such as

```
[[al:num]]
```

or

```
[[:munla]]
```

<> at require-statement should be quotes
(F) You wrote `require <file>` when you should have written `require 'file'`.

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 freed package
(F) You wrote `bless $foo` with one argument after somehow causing the current package to be freed. Perl cannot figure out what to do, so it throws up in hands in despair.

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 clear deleted array
(S debugging) An array was assigned to when it was being freed. Freed values are not supposed to be visible to Perl code. This can also happen if XS code calls `av_clear` from a custom magic callback on the array.

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%x
(S 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 '%s'%s
(S 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: SV 0x%x
(S 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
(S internal) The reference counts got screwed up on symbol aliases.

Attempt to free unreferenced scalar: SV 0x%x
(S 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()* w as 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 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(1) and "%INC" in perlvar.

Attempt to set length of freed array
(W misc) 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.

Attribute "locked" is deprecated
(D deprecated) You have used the attributes pragma to modify the "locked" attribute on a code reference. The :locked attribute is obsolete, has had no effect since 5005 threads were removed, and will be removed in a future release of Perl 5.

Attribute prototype(%s) discards earlier prototype attribute in same sub
(W misc) A sub was declared as sub foo : prototype(A) : prototype(B) {}, for example. Since each sub can only have one prototype, the earlier declaration(s) are discarded while the last one is applied.

Attribute ‘‘unique’’ is deprecated

(D deprecated) You have used the attributes pragma to modify the ‘‘unique’’ attribute on an array, hash or scalar reference. The :unique attribute has had no effect since Perl 5.8.8, and will be removed in a future release of Perl 5.

av_reify called on tied array

(S debugging) This indicates that something went wrong and Perl got *very* confused about `@_` or `@DB::args` being tied.

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

(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(structmsqid_ds*), sizeof(structsemid_ds*), and sizeof(structshmid_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 plugin affecting keyword ’%s’

(F) An extension using the keyword plugin mechanism violated the plugin API.

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 the 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.

Bad symbol for scalar
(P) An internal request asked to add a scalar 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.

\%d better written as $%d
(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(1) 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 your 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`
(P) Perl detected an attempt to copy an internal value that is not copiable.

Bizarre SvTYPE [%d]

(P) When starting a new thread or returning values from a thread, Perl encountered an invalid data type.

Both or neither range ends should be Unicode in regex; marked by <--HERE in m/%s/

(W regexp) (only under `usere'strict'` or within `(?[...])`)

In a bracketed character class in a regular expression pattern, you had a range which has exactly one end of it specified using `\N{}`, and the other end is specified using a non-portable mechanism. Perl treats the range as a Unicode range, that is, all the characters in it are considered to be the Unicode characters, and which may be different code points on some platforms Perl runs on. For example, `[\N{U+06}-\x08]` is treated as if you had instead said `[\N{U+06}-\N{U+08}]`, that is it matches the characters whose code points in Unicode are 6, 7, and 8. But that `\x08` might indicate that you meant something different, so the warning gets raised.

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.

Calling POSIX::%s() is deprecated

(D deprecated) You called a function whose use is deprecated. See the function's name in POSIX for details.

Cannot chr `%f`

(F) You passed an invalid number (like an infinity or not-a-number) to `chr`.

Cannot compress `%f` in pack

(F) You tried compressing an infinity or not-a-number as an unsigned integer with BER, which makes no sense.

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 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`

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

Cannot find encoding "%s"

(S io) You tried to apply an encoding that did not exist to a filehandle, either with *open()* or *binmode()*.

Cannot pack %f with '%c'
(F) You tried converting an infinity or not-a-number to an integer, which makes no sense.

Cannot printf %f with '%c'
(F) You tried printing an infinity or not-a-number as a character (%c), which makes no sense. Maybe you meant '%s', or just stringifying it?

Cannot set tied @DB::args
(F) caller tried to set @DB::args, but found it tied. Tying @DB::args is not supported. (Before this error was added, it used to crash.)

Cannot tie unreifiable array
(P) You somehow managed to call tie on an array that does not keep a reference count on its arguments and cannot be made to do so. Such arrays are not even supposed to be accessible to Perl code, but are only used internally.

Cannot yet reorder *sv_catpvfn()* arguments from va_list
(F) Some XS code tried to use sv_catpvfn() or a related function with a format string that specifies explicit indexes for some of the elements, and using a C-style variable-argument list (a va_list). This is not currently supported. XS authors wanting to do this must instead construct a C array of SV* scalars containing the arguments.

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" 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 call *mro_isa_changed_in()* on anonymous symbol table
(P) Perl got confused as to whether a hash was a plain hash or a symbol table hash when trying to update @ISA caches.

Can't call *mro_method_changed_in()* on anonymous symbol table
(F) An XS module tried to call `mro_method_changed_in` on a hash that was not attached to the symbol table.

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 `%s` to `%s` 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 "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 `%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 "default" outside a topicalizer
(F) You have used a `default` block that is neither inside a `foreach` loop nor a `given` block. (Note that this error is issued on exit from the `default` block, so you won't get the error if you use an explicit `continue`.)

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, a FIFO or an uneditable directory. 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 %s("%s") on non-UTF-8 locale; resolved to "%s".
(W locale) You are 1) running under "`use locale`"; 2) the current locale is not a UTF-8 one; 3) you tried to do the designated case-change operation on the specified Unicode character; and 4) the result of this operation would mix Unicode and locale rules, which likely conflict. Mixing of different rule types is forbidden, so the operation was not done; instead the result is the indicated value, which is the best available that uses entirely Unicode rules. That turns out to almost always be the original character, unchanged.

It is generally a bad idea to mix non-UTF-8 locales and Unicode, and this issue is one of the reasons

why. This warning is raised when Unicode rules would normally cause the result of this operation to contain a character that is in the range specified by the locale, 0..255, and hence is subject to the locale's rules, not Unicode's.

If you are using locale purely for its characteristics related to things like its numeric and time formatting (and not `LC_CTYPE`), consider using a restricted form of the locale pragma (see ''The ''use locale'' pragma'' in perllocale) like "`uselocale':not_characters'`".

Note that failed case-changing operations done as a result of case-insensitive `/i` regular expression matching will show up in this warning as having the `fc` operation (as that is what the regular expression engine calls behind the scenes.)

Can't do waitpid with flags
(F) This machine doesn't have either *waitpid()* or *wait4()*, so only *waitpid()* without flags is emulated.

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 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 or there may not be a linebreak after it. A good programmer's editor will have a way to help you find these characters (or lack of characters). See perlop(1) for the full details on here-

documents.

Can't find Unicode property definition "%s"

Can't find Unicode property definition "%s" in regex; marked by <-- HERE in m/%s/

(F) The named property which you specified via `\p` or `\P` is not one known to Perl. Perhaps you misspelled the name? See "Properties accessible through \p{} and \P{}" in perluniprops(1) for a complete list of available official properties. If it is a user-defined property it must have been defined by the time the regular expression is matched.

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, or until `\E`).

Can't fork: `%s`

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

Can't fork, trying again in 5 seconds

(W pipe) A fork in a piped open failed with EAGAIN and will be retried after five seconds.

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 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 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 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 careless.

Can't kill a non-numeric process ID
(F) Process identifiers must be (signed) integers. It is a fatal error to attempt to *kill()* an undefined, empty-string or otherwise non-numeric process identifier.

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(1) 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 object method ''%s'' via package ''%s'' (perhaps you forgot to load ''%s''?)
(F) You called a method on a class that did not exist, and the method could not be found in UNIVERSAL. This often means that a method requires a package that has not been loaded.

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 make loaded symbols global on this platform while loading `%s`
(S) A module passed the flag 0x01 to *DynaLoader::dl_load_file()* to request that symbols from the stated file are made available globally within the process, but that functionality is not available on this platform. Whilst the module likely will still work, this may prevent the perl interpreter from loading other XS-based extensions which need to link directly to functions defined in the C or XS code in the stated file.

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 of &%s
(F) Subroutines meant to be used in lvalue context should be declared as such. See "Lvalue subroutines" in perlsub.

Can't modify reference to `%s` in `%s` assignment
(F) Only a limited number of constructs can be used as the argument to a reference constructor on the left-hand side of an assignment, and what you used was not one of them. See "Assigning to References" in perlref.

Can't modify reference to localized parenthesized array in list assignment
(F) Assigning to `\local(@array)` or `\(local @array)` is not supported, as it is not clear exactly what it should do. If you meant to make `@array` refer to some other array, use `\@array = \@other_array`. If you want to make the elements of `@array` aliases of the scalars referenced on the right-hand side, use `\(@array) = @scalar_refs`.

Can't modify reference to parenthesized hash in list assignment
(F) Assigning to `\(%hash)` is not supported. If you meant to make `%hash` refer to some other hash, use `\%hash = \%other_hash`. If you want to make the elements of `%hash` into aliases of the scalars referenced on the right-hand side, use a hash slice: `\@hash{@keys} = @those_scalar_refs`.

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.

(F) You tried to call perl with the **-e** switch, but */dev/null* (or your operating system's equivalent) could not be opened.

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”: `%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 CR TL’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 redeclare “%s” in “%s”
(F) A “my”, “our” or “state” declaration was found within another declaration, such as `my ($x, my($y), $z)`or `our (my $x)`.

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 represent character for Ox%X on this platform
(F) There is a hard limit to how big a character code point can be due to the fundamental properties of UTF-8, especially on EBCDIC platforms. The given code point exceeds that. The only work-around is

to not use such a large code point.

Can't reset %ENV on this system
(F) You called `reset('E')` or similar, which tried to reset all variables in the current package beginning with "E". In the main package, that includes %ENV. Resetting %ENV is not supported on some systems, notably VMS.

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 the method name is ???, this is an internal error.

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 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 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 '%c' after -mname
(F) You tried to call perl with the **-m** switch, but you put something other than "=" after the module name.

Can't use a hash as a reference
(F) You tried to use a hash as a reference, as in `%foo->{"bar"}` or `%$ref->{"hello"}`. Versions of perl <= 5.22.0 used to allow this syntax, but shouldn't have. This was deprecated in perl 5.6.1.

Can't use an array as a reference
(F) You tried to use an array as a reference, as in `@foo->[23]` or `@$ref->[99]`. Versions of perl <= 5.22.0 used to allow this syntax, but shouldn't have. This was deprecated in perl 5.6.1.

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 'defined(@array)' (Maybe you should just omit the *defined()*?)
(F) *defined()* is not 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.

Can't use 'defined(%hash)' (Maybe you should just omit the *defined()*?)
(F) `defined()` is not usually right on hashes.

Although `defined %hash` is false on a plain not-yet-used hash, it becomes true in several non-obvious circumstances, including iterators, weak references, stash names, even remaining true after `undef %hash`. These things make `defined %hash` fairly useless in practice, so it now generates a fatal error.

If a check for non-empty is what you wanted then just put it in boolean context (see ''Scalar values'' in perldata):

```
if (%hash) {
# not empty
}
```

If you had `defined %Foo::Bar::QUUX` to check whether such a package variable exists then that's never really been reliable, and isn't a good way to enquire about the features of a package, or whether it's loaded, etc.

Can't use %s for loop variable
(P) The parser got confused when trying to parse a `foreach` loop.

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 weren't.

Can't use '%c' 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

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

(F) You've told Perl to dereference a string, something which `use strict` blocks to prevent it happening accidentally. See "Symbolic references" in perlref. This can be triggered by an @ or $ in a double-quoted string immediately before interpolating a variable, for example in `"user @$twitter_id"`, which says to treat the contents of `$twitter_id` as an array reference; use a \ to have a literal @ symbol followed by the contents of `$twitter_id`: `"user \@$twitter_id"`.

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 weaken a nonreference

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

Can't "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 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 following "\c" must be printable ASCII

(F) In `\c`*X*, *X* must be a printable (non-control) ASCII character.

Note that ASCII characters that don't map to control characters are discouraged, and will generate the warning (when enabled) ""\c%c" is more clearly written simply as "%s"".

Character following \%c must be '{' or a single-character Unicode property name in regex; marked by <-- HERE in m/%s/

(F) (In the above the `%c` is replaced by either `p` or `P`.) You specified something that isn't a legal Unicode property name. Most Unicode properties are specified by `\p{...}`. But if the name is a single character one, the braces may be omitted.

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 '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 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(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")
```

charnames alias definitions may not contain a sequence of multiple spaces
(F) You defined a character name which had multiple space characters in a row. Change them to single spaces. Usually these names are defined in the `:alias` import argument to `use charnames`, but they could be defined by a translator installed into `$^H{charnames}`. See "CUSTOM ALIASES" in charnames.

charnames alias definitions may not contain trailing white-space
(F) You defined a character name which ended in a space character. Remove the trailing space(s). Usually these names are defined in the `:alias` import argument to `use charnames`, but they could be defined by a translator installed into `$^H{charnames}`. See "CUSTOM ALIASES" in charnames.

chdir() on unopened filehandle `%s`
(W unopened) You tried *chdir()* on a filehandle that was never opened.

"\c%c" is more clearly written simply as "%s"
(W syntax) The `\c`*X* construct is intended to be a way to specify non-printable characters. You used it for a printable one, which is better written as simply itself, perhaps preceded by a backslash for non-word characters. Doing it the way you did is not portable between ASCII and EBCDIC platforms.

Cloning substitution context is unimplemented
(F) Creating a new thread inside the `s///` operator is not supported.

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.

close() on unopened filehandle `%s`
(W unopened) You tried to close a filehandle that was never opened.

Closure prototype called
(F) If a closure has attributes, the subroutine passed to an attribute handler is the prototype that is cloned when a new closure is created. This subroutine cannot be called.

\C no longer supported in regex; marked by <--HERE in m/%s/
(F) The \C character class used to allow a match of single byte within a multi-byte utf-8 character, but was removed in v5.24 as it broke encapsulation and its implementation was extremely buggy. If you really need to process the individual bytes, you probably want to convert your string to one where each underlying byte is stored as a character, with *utf8::encode()*.

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.

Code point 0x%X is not Unicode, and not portable
(S non_unicode) You had a code point that has never been in any standard, so it is likely that languages other than Perl will NOT understand it. At one time, it was legal in some standards to have code points up to 0x7FFF_FFFF, but not higher, and this code point is higher.

Acceptance of these code points is a Perl extension, and you should expect that nothing other than Perl can handle them; Perl itself on EBCDIC platforms before v5.24 does not handle them.

Code points above 0xFFFF_FFFF require larger than a 32 bit word.

Perl also makes no guarantees that the representation of these code points won't change at some point in the future, say when machines become available that have larger than a 64-bit word. At that time, files written by an older Perl would require conversion before being readable by a newer Perl.

Code point 0x%X is not Unicode, may not be portable
(S non_unicode) You had a code point above the Unicode maximum of U+10FFFF.

Perl allows strings to contain a superset of Unicode code points, but these may not be accepted by other languages/systems. Further, even if these languages/systems accept these large code points, they may have chosen a different representation for them than the UTF-8-like one that Perl has, which would mean files are not exchangeable between them and Perl.

On EBCDIC platforms, code points above 0x3FFF_FFFF have a different representation in Perl v5.24 than before, so any file containing these that was written before that version will require conversion before being readable by a later Perl.

`%s:` Command not found
(A) You've accidentally run your script through **csh** or another shell instead of Perl. Check the #! line, or manually feed your script into Perl yourself. The #! line at the top of your file could look like

```
#!/usr/bin/perl -w
```

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(1) for information on *Mastering Regular Expressions*.)

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): Call to &{$^H{%s}} did not return a defined value

(F) The subroutine registered to handle constant overloading (see overload) or a custom charnames handler (see “CUSTOM TRANSLATORS” in charnames) returned an undefined value.

Constant(%s): $^H{%s} is not defined

(F) The parser found inconsistencies while attempting to define an overloaded constant. Perhaps you forgot to load the corresponding overload pragma?

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(1) and constant.

Constants from lexical variables potentially modified elsewhere are deprecated

(D deprecated) You wrote something like

```
my $var;
$sub = sub () { $var };
```

but `$var` is referenced elsewhere and could be modified after the `sub` expression is evaluated. Either it is explicitly modified elsewhere (`$var = 3`) or it is passed to a subroutine or to an operator like `printf` or `map`, which may or may not modify the variable.

Traditionally, Perl has captured the value of the variable at that point and turned the subroutine into a constant eligible for inlining. In those cases where the variable can be modified elsewhere, this breaks the behavior of closures, in which the subroutine captures the variable itself, rather than its value, so future changes to the variable are reflected in the subroutine’s return value.

This usage is deprecated, because the behavior is likely to change in a future version of Perl.

If you intended for the subroutine to be eligible for inlining, then make sure the variable is not referenced elsewhere, possibly by copying it:

```
my $var2 = $var;
$sub = sub () { $var2 };
```

If you do want this subroutine to be a closure that reflects future changes to the variable that it closes over, add an explicit `return`:

```
my $var;
$sub = sub () { return $var };
```

Constant subroutine `%s` redefined

(W redefine)(S) You redefined a subroutine which had previously been eligible for inlining. See “Constant Functions” in perlsub(1) 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(1) for commentary and workarounds.

Constant(%s) unknown

(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 pragma?

:const is experimental
(S experimental::const_attr) The ‘‘const’’ attribute is experimental. If you want to use the feature, disable the warning with `no warnings 'experimental::const_attr'` but know that in doing so you are taking the risk that your code may break in a future Perl version.

:const is not permitted on named subroutines
(F) The ‘‘const’’ attribute causes an anonymous subroutine to be run and its value captured at the time that it is cloned. Named subroutines are not cloned like this, so the attribute does not make sense on them.

Copy method did not return a reference
(F) The method which overloads ‘‘=’’ is buggy. See ‘‘Copy Constructor’’ in overload.

&CORE::%s cannot be called directly
(F) You tried to call a subroutine in the `CORE::` namespace with `&foo` syntax or through a reference. Some subroutines in this package cannot yet be called that way, but must be called as barewords. Something like this will work:

```
BEGIN { *shove = \&CORE::push; }
shove @array, 1,2,3; # pushes on to @array
```

CORE::%s is not a keyword
(F) The CORE:: namespace is reserved for Perl keywords.

Corrupted regexp opcode `%d` > `%d`
(P) This is either an error in Perl, or, if you’re using one, your custom regular expression engine. If not the latter, report the problem through the perlbug(1) utility.

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%x at 0x%x
(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 anonymous subroutine
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.

(?(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 whereabouts in the regular expression the problem was discovered.

`%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`.

delete argument is index/value array slice, use array slice
(F) You used index/value array slice syntax (`%array[...]`) as the argument to `delete`. You probably meant `@array[...]` with an @ symbol instead.

delete argument is key/value hash slice, use hash slice
(F) You used key/value hash slice syntax (`%hash{...}`) as the argument to `delete`. You probably meant `@hash{...}` with an @ symbol instead.

delete 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"}
```

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 5.10.0, 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 misc) 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 `$@` was 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 get file name
: (P) `PerlIO_getname`, a perl internal I/O function specific to VMS, was somehow called on another platform. This should not happen.

Don’t know how to handle magic of type \%o
: (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 unpack) 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 \%c in regex; marked by <--HERE in m/%s/
Empty \%c{} in regex; marked by <--HERE in m/%s/
: (F) `\p` and `\P` are used to introduce a named Unicode property, as described in perlunicode(1) 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.

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(1), and perlsec.

Eval-group not allowed at runtime, use re 'eval' in regex m/%s/
: (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 using the `re 'eval'` pragma or by explicitly building the pattern from an interpolated string at run time and using that in an *eval()*. See "(?{ code })" in perlre.

Eval-group not allowed, use re 'eval' in regex m/%s/
: (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 whereabouts in the regular expression 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 on some systems, e.g., Symbian OS. See perlport.

Execution of %s aborted due to compilation errors.
: (F) The final summary message when a Perl compilation fails.

exists 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
```

exists 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.

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.

Expecting close bracket in regex; marked by <--HERE in m/%s/
(F) You wrote something like

```
(?13
```

to denote a capturing group of the form `(?PARNO)`, but omitted the `")"`.

Expecting '(?flags:(?[...' in regex; marked by <--HERE in m/%s/
(F) The `(?[...])` extended character class regular expression construct only allows character classes (including character class escapes like `\d`), operators, and parentheses. The one exception is `(?flags:...)` containing at least one flag and exactly one `(?[...])` construct. This allows a regular expression containing just `(?[...])` to be interpolated. If you see this error message, then you probably have some other `(?...)` construct inside your character class. See ''Extended Bracketed Character Classes'' in perlrecharclass.

Experimental aliasing via reference not enabled
(F) To do aliasing via references, you must first enable the feature:

```
no warnings "experimental::refaliasing";
use feature "refaliasing";
\$x = \$y;
```

Experimental %s on scalar is now forbidden
(F) An experimental feature added in Perl 5.14 allowed `each`, `keys`, `push`, `pop`, `shift`, `splice`, `unshift`, and `values` to be called with a scalar argument. This experiment is considered unsuccessful, and has been removed. The `postderef` feature may meet your needs better.

Experimental subroutine signatures not enabled
(F) To use subroutine signatures, you must first enable them:

```
no warnings "experimental::signatures";
use feature "signatures";
sub foo ($left, $right) { ... }
```

Experimental ''%s'' subs not enabled
(F) To use lexical subs, you must first enable them:

```
no warnings 'experimental::lexical_subs';
use feature 'lexical_subs';
my sub foo { ... }
```

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.

Failed to close in-place edit file `%s: %s`

(F) Closing an output file from in-place editing, as with the `-i` command-line switch, failed.

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

(W regexp)(F) 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 ‘‘-’’. In a `(?[...])` construct, this is an error, rather than a warning. Consider quoting the ‘‘-’’, ‘‘\-’’. The <--HERE shows whereabouts in the regular expression the problem was discovered. See perlre.

Fatal VMS error (status=%d) 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 starts 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 the 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 ‘‘>’’. 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.

given is experimental
(S experimental::smartmatch) `given` depends on smartmatch, which is experimental, so its behavior may change or even be removed in any future release of perl. See the explanation under ‘‘Experimental Details on given and when’’ in perlsyn.

Global symbol ‘‘%s’’ requires explicit package name (did you forget to declare ‘‘my %s’’?)
(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)
(S 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 reb uild 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’’.

gmtime(%f) failed
(W overflow) You called `gmtime` with a number that it could not handle: too large, too small, or NaN. The returned value is `undef`.

gmtime(%f) too large
(W overflow) You called `gmtime` with a number that was larger than it can reliably handle and `gmtime` probably returned the wrong date. This warning is also triggered with NaN (the special not-a-number value).

gmtime(%f) too small
(W overflow) You called `gmtime` with a number that was smaller than it can reliably handle and `gmtime` probably returned the wrong date.

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.

Goto undefined subroutine%s
(F) You tried to call a subroutine with `goto &sub` syntax, but the indicated subroutine hasn’t been defined, or if it was, it has since been undefined.

Group name must start with a non-digit word character in regex; marked by <--HERE in m/%s/
(F) Group names must follow the rules for perl identifiers, meaning they must start with a non-digit word character. A common cause of this error is using (?&0) instead of (?0). See perlre.

()-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.

`%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.

Having more than one /%c regexp modifier is deprecated
(D deprecated, regexp) You used the indicated regular expression pattern modifier at least twice in a string of modifiers. It is deprecated to do this with this particular modifier, to allow future extensions to the Perl language.

Hexadecimal float: exponent overflow
(W overflow) The hexadecimal floating point has a larger exponent than the floating point supports.

Hexadecimal float: exponent underflow
(W overflow) The hexadecimal floating point has a smaller exponent than the floating point supports.

Hexadecimal float: internal error (%s)
(F) Something went horribly bad in hexadecimal float handling.

Hexadecimal float: mantissa overflow
(W overflow) The hexadecimal floating point literal had more bits in the mantissa (the part between the 0x and the exponent, also known as the fraction or the significand) than the floating point supports.

Hexadecimal float: precision loss
(W overflow) The hexadecimal floating point had internally more digits than could be output. This can be caused by unsupported long double formats, or by 64-bit integers not being available (needed to retrieve the digits under some configurations).

Hexadecimal float: unsupported long double format
(F) You have configured Perl to use long doubles but the internals of the long double format are unknown; therefore the hexadecimal float output is impossible.

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(1) 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 zero length \N{} in character class in regex; marked by <--HERE in m/%s/
(W regexp) Named Unicode character escapes (`\N{...}`) may return a zero-length sequence. When such an escape is used in a character class its behavior 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 after '_' in prototype for `%s` : `%s`
(W illegalproto) An illegal character was found in a prototype declaration. The '_' in a prototype must be followed by a ';', indicating the rest of the parameters are optional, or one of '@' or '%', since those two will accept 0 or more final parameters.

Illegal character \%o (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 illegalproto) An illegal character was found in a prototype declaration. Legal characters in prototypes are $, @, %, *, ;, [,], &, \, and +. Perhaps you were trying to write a subroutine signature but didn't enable that feature first (`use feature 'signatures'`), so your signature was instead interpreted as a bad prototype.

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 pattern in regex; marked by <--HERE in m/%s/
(F) You wrote something like

```
(?+foo)
```

The `"+"` is valid only when followed by digits, indicating a capturing group. See *(?PARNO)*.

Illegal suidscript
(F) The script run under suidperl was somehow illegal.

Illegal switch in PERL5OPT: -%c
(X) The PERL5OPT environment variable may only be used to set the following switches: **-[CDIMUdmtw]**.

Illegal user-defined property name
(F) You specified a Unicode-like property name in a regular expression pattern (using `\p{}` or `\P{}`) that Perl knows isn't an official Unicode property, and was likely meant to be a user-defined property name, but it can't be one of those, as they must begin with either `In` or `Is`. Check the spelling. See also ''Can't find Unicode property definition ''%s"".

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.

Incomplete expression within '(?[ ])' in regex; marked by <--HERE in m/%s/
(F) There was a syntax error within the `(?[ ])`. This can happen if the expression inside the construct was completely empty, or if there are too many or few operands for the number of operators. Perl is not smart enough to give you a more precise indication as to what is wrong.

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.

Infinite recursion in regex
(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.

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.

%%s[%s] in scalar context better written as $%s[%s]
(W syntax) In scalar context, you've used an array index/value 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 in the value it returns and when evaluating its argument, while `%foo[&bar]` provides a list context to its subscript, which can do weird things if you're expecting only one subscript. When called in list context, it also returns the index (what `&bar` returns) in addition to the value.

%%s{%s} in scalar context better written as $%s{%s}
(W syntax) In scalar context, you've used a hash key/value 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 in the value it returns and when evaluating its argument, while `@foo{&bar}` and provides a list context to its subscript, which can do weird things if you're expecting only one subscript. When called in list context, it also returns the key in addition to the value.

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(1) 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.

Insecure user-defined property `%s`
(F) Perl detected tainted data when trying to compile a regular expression that contains a call to a user-defined character property function, i.e. `\p{IsFoo}` or `\p{InFoo}`. See ''User-Defined Character Properties'' in perlunicode(1) and perlsec.

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 `%s` number
(S 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 srand

(S overflow) The number you have passed to srand is too big to fit in your architecture's integer representation. The number has been replaced with the largest integer supported (0xFFFFFFFF on 32-bit architectures). This means you may be getting less randomness than you expect, because different random seeds above the maximum will return the same sequence of random numbers.

Integer overflow in version

Integer overflow in version %d

(W overflow) 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 an 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 whereabouts in the regular expression 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 %<num>p might conflict with future printf extensions

(S internal) Perl's internal routine that handles `printf` and `sprintf` formatting follows a slightly different set of rules when called from C or XS code. Specifically, formats consisting of digits followed by ''p'' (e.g., ''%7p'') are reserved for future use. If you see this message, then an XS module tried to call that routine with one such reserved format.

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

(P) Something went badly awry in the regular expression parser. The <--HERE shows whereabouts in the regular expression 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.

In '(?...)', the '(' and '?' must be adjacent in regex; marked by <--HERE in m/%s/

(F) The two-character sequence `"(?"` in this context in a regular expression pattern should be an indivisible token, with nothing intervening between the `"("` and the `"?"`, but you separated them with whitespace.

Invalid %s attribute: %s

(F) 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

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

Invalid character in charnames alias definition; marked by <--HERE in '%s

(F) You tried to create a custom alias for a character name, with the `:alias` option to `use charnames` and the specified character in the indicated name isn't valid. See ''CUSTOM ALIASES'' in charnames.

Invalid \0 character in %s for %s: %s\0%s

(W syscalls) Embedded \0 characters in pathnames or other system call arguments produce a warning as of 5.20. The parts after the \0 were formerly ignored by system calls.

Invalid character in \N{...}; marked by <--HERE in \N{%s}
(F) Only certain characters are valid for character names. The indicated one isn't. See ''CUSTOM ALIASES'' in charnames.

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)(F) 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, except within `(?[])`, where it is a fatal error. The <--HERE shows whereabouts in the regular expression the escape was discovered.

Invalid hexadecimal number in \N{U+...}
Invalid hexadecimal number in \N{U+...} in regex; marked by <--HERE in m/%s/
(F) The character constant represented by `...` is not a valid hexadecimal number. Either it is empty, or you tried to use a character other than 0 - 9 or A - F, a - f in a hexadecimal number.

Invalid module name `%s` with -%c option: contains single ':'
(F) The module argument to perl's **-m** and **-M** command-line options cannot contain single colons in the module name, but only in the arguments after ''='' . In other words, **-MFoo::Bar=:baz** is ok, but **-MFoo:Bar=baz** is not.

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 supported are `dfs` and `c3`, unless you have loaded a module that is a MRO plugin. See mro and perlmroapi.

Invalid negative number (%s) in chr
(W utf8) You passed a negative number to `chr`. Negative numbers are not valid character numbers, so it returns the Unicode replacement character (U+FFFD).

Invalid number '%s' for -C option.
(F) You supplied a number to the -C option that either has extra leading zeroes or overflows perl's unsigned integer representation.

invalid option -D%c, use -D'' to see choices
(S debugging) Perl was called with invalid debugger flags. Call perl with the **-D** option with no flags to see the list of acceptable values. See also ''-Dletters'' in perlrun.

Invalid quantifier in {,} in regex; marked by <--HERE in m/%s/
(F) The pattern looks like a {min,max} quantifier, but the min or max could not be parsed as a valid number - either it has leading zeroes, or it represents too big a number to cope with. The <--HERE shows where in the regular expression the problem was discovered. See perlre.

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 whereabouts in the regular expression 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 strict version format (%s)
(F) A version number did not meet the ''strict'' criteria for versions. A ''strict'' version number is a positive decimal number (integer or decimal-fraction) without exponentiation or else a dotted-decimal v-string with a leading 'v' character and at least three components. The parenthesized text indicates which criteria were not met. See the version module for more details on allowed version formats.

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 (%s)
(F) A version number did not meet the ''lax'' criteria for versions. A ''lax'' version number is a positive decimal number (integer or decimal-fraction) without exponentiation or else a dotted-decimal v-string. If the v-string has fewer than three components, it must have a leading 'v' character. Otherwise, the leading 'v' is optional. Both decimal and dotted-decimal versions may have a trailing ''alpha'' component separated by an underscore character after a fractional or dotted-decimal component. The parenthesized text indicates which criteria were not met. See the version module for more details on allowed version formats.

Invalid version object
(F) The internal structure of the version object was invalid. Perhaps the internals were modified directly in some way or an arbitrary reference was blessed into the ''version'' class.

In '(*VERB...)', the '(' and '*' must be adjacent in regex; marked by <--HERE in m/%s/
(F) The two-character sequence `"(*"` in this context in a regular expression pattern should be an indivisible token, with nothing intervening between the `"("` and the `"*"`, but you separated them.

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 your 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 nor an ioctl call (SIOCATMARK).

'%s' is an unknown bound type in regex; marked by <--HERE in m/%s/
(F) You used `\b{...}` or `\B{...}` and the `...` is not known to Perl. The current valid ones are given in ''\b{}, \b, \B{}, \B'' in perlrebackslash.

%s() is deprecated on :utf8 handles
(W deprecated) The *sysread()*, *recv()*, *syswrite()* and *send()* operators are deprecated on handles that have the `:utf8` layer, either explicitly, or implicitly, eg., with the `:encoding(UTF-16LE)` layer.

Both *sysread()* and *recv()* currently use only the `:utf8` flag for the stream, ignoring the actual layers. Since *sysread()* and *recv()* do no UTF-8 validation they can end up creating invalidly encoded scalars.

Similarly, *syswrite()* and *send()* use only the `:utf8` flag, otherwise ignoring any layers. If the flag is set, both write the value UTF-8 encoded, even if the layer is some different encoding, such as the example above.

Ideally, all of these operators would completely ignore the `:utf8` state, working only with bytes, but this would result in silently breaking existing code. To avoid this a future version of perl will throw an exception when any of *sysread()*, *recv()*, *syswrite()* or *send()* are called on handle with the `:utf8` layer.

‘‘%s’’ is more clearly written simply as ‘‘%s’’ in regex; marked by <--HERE in m/%s/
(W regexp) (only under `usere'strict'` or within `(?[...])`)

You specified a character that has the given plainer way of writing it, and which is also portable to platforms running with different character sets.

$* is no longer supported
(D deprecated, syntax) The special variable `$*`, deprecated in older perls, has been removed as of 5.10.0 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 `$*` you should use the `/m` (and maybe `/s`) regexp modifiers. You can enable `/m` for a lexical scope (even a whole file) with `use re '/m'`. (In older versions: when `$*` was set to a true value then all regular expressions behaved as if they were written using `/m`.)

$# is no longer supported
(D deprecated, syntax) The special variable `$#`, deprecated in older perls, has been removed as of 5.10.0 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.

-i used with no filenames on the command line, reading from STDIN
(S inplace) The `-i` option was passed on the command line, indicating that the script is intended to edit files in place, but no files were given. This is usually a mistake, since editing STDIN in place doesn’t make sense, and can be confusing because it can make perl look like it is hanging when it is really just trying to read from STDIN. You should either pass a filename to edit, or remove `-i` from the command line. See perlrun(1) for more details.

Junk on end of regexp in regex m/%s/
(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.

length() used on `%s` (did you mean ‘‘scalar(%s)’’?)
(W syntax) You used *length()* on either an array or a hash when you probably wanted a count of the items.

Array size can be obtained by doing:

```
scalar(@array);
```

The number of items in a hash can be obtained by doing:

```
scalar(keys %hash);
```

Lexing code attempted to stuff non-Latin-1 character into Latin-1 input
(F) An extension is attempting to insert text into the current parse (using lex_stuff_pvn or similar), but tried to insert a character that couldn't be part of the current input. This is an inherent pitfall of the stuffing mechanism, and one of the reasons to avoid it. Where it is necessary to stuff, stuffing only plain ASCII is recommended.

Lexing code internal error (%s)
(F) Lexing code supplied by an extension violated the lexer's API in a detectable way.

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.

List form of piped open not implemented
(F) On some platforms, notably Windows, the three-or-more-arguments form of `open` does not support pipes, such as `open($pipe, '|-', @args)`. Use the two-argument `open($pipe, '|prog arg1 arg2...')` form instead.

%s: loadable library and perl binaries are mismatched (got handshake key %p, needed %p)
(P) A dynamic loading library `.so` or `.dll` was being loaded into the process that was built against a different build of perl than the said library was compiled against. Reinstalling the XS module will likely fix this error.

Locale '%s' may not work well.%s
(W locale) You are using the named locale, which is a non-UTF-8 one, and which perl has determined is not fully compatible with what it can handle. The second %s gives a reason.

By far the most common reason is that the locale has characters in it that are represented by more than one byte. The only such locales that Perl can handle are the UTF-8 locales. Most likely the specified locale is a non-UTF-8 one for an East Asian language such as Chinese or Japanese. If the locale is a superset of ASCII, the ASCII portion of it may work in Perl.

Some essentially obsolete locales that aren't supersets of ASCII, mainly those in ISO 646 or other 7-bit locales, such as ASMO 449, can also have problems, depending on what portions of the ASCII character set get changed by the locale and are also used by the program. The warning message lists the determinable conflicting characters.

Note that not all incompatibilities are found.

If this happens to you, there's not much you can do except switch to use a different locale or use Encode to translate from the locale into UTF-8; if that's impracticable, you have been warned that some things may break.

This message is output once each time a bad locale is switched into within the scope of `uselocale`, or on the first possibly-affected operation if the `uselocale` inherits a bad one. It is not raised for any operations from the POSIX module.

localtime(%f) failed
(W overflow) You called `localtime` with a number that it could not handle: too large, too small, or NaN. The returned value is `undef`.

localtime(%f) too large
(W overflow) You called `localtime` with a number that was larger than it can reliably handle and `localtime` probably returned the wrong date. This warning is also triggered with NaN (the special not-a-number value).

localtime(%f) too small
(W overflow) You called `localtime` with a number that was smaller than it can reliably handle and `localtime` probably returned the wrong date.

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.

Lost precision when `%s` `%f` by 1
(W imprecision) The value you attempted to increment or decrement by one is too large for the underlying floating point representation to store accurately, hence the target of `++` or `--` is unchanged. Perl issues this warning because it has already switched from integers to floating point when values are too large for integers, and now even floating point is insufficient. You may wish to switch to using Math::BigInt explicitly.

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 attribute `%s` already-defined subroutine
(W misc) Although attributes.pm allows this, turning the lvalue attribute on or off on a Perl subroutine that is already defined does not always work properly. It may or may not do what you want, depending on what code is inside the subroutine, with exact details subject to change between Perl versions. Only do this if you really know what you are doing.

lvalue attribute ignored after the subroutine has been defined
(W misc) Using the `:lvalue` declarative syntax to make a Perl subroutine an lvalue subroutine after it has been defined is not permitted. To make the subroutine an lvalue subroutine, add the lvalue attribute to the definition, or put the `sub foo :lvalue;` declaration before the definition.

See also attributes.pm.

Magical list constants are not supported
(F) You assigned a magical array to a stash element, and then tried to use the subroutine from the same slot. You are asking Perl to do something it cannot do, details subject to change between Perl versions.

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. Perhaps the function’s author was trying to write a subroutine signature but didn’t enable that feature first (`use feature 'signatures'`), so the signature was instead interpreted as a bad prototype.

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-8 character immediately after '%s'
(F) You said `use utf8`, but the program file doesn't comply with UTF-8 encoding rules. The message prints out the properly encoded characters just before the first bad one. If `utf8` warnings are enabled, a warning is generated that gives more details about the type of malformation.

Malformed UTF-8 returned by \N{%s} immediately after '%s'
(F) The charnames handler returned malformed UTF-8.

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.

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

Mandatory parameter follows optional parameter
(F) In a subroutine signature, you wrote something like ''$a = undef, `$b`'', making an earlier parameter optional and a later one mandatory. Parameters are filled from left to right, so it's impossible for the caller to omit an earlier one and pass a later one. If you want to act as if the parameters are filled from right to left, declare the rightmost optional and then shuffle the parameters around in the subroutine's body.

Matched non-Unicode code point 0x%X against Unicode property; may not be portable
(S non_unicode) Perl allows strings to contain a superset of Unicode code points; each code point may be as large as what is storable in an unsigned integer on your system, but these may not be accepted by other languages/systems. This message occurs when you matched a string containing such a code point against a regular expression pattern, and the code point was matched against a Unicode property, `\p{...}` or `\P{...}`. Unicode properties are only defined on Unicode code points, so the result of this match is undefined by Unicode, but Perl (starting in v5.20) treats non-Unicode code points as if they were typical unassigned Unicode ones, and matched this one accordingly. Whether a given property matches these code points or not is specified in ''Properties accessible through \p{} and \P{}'' in perluniprops.

This message is suppressed (unless it has been made fatal) if it is immaterial to the results of the match if the code point is Unicode or not. For example, the property `\p{ASCII_Hex_Digit}` only can match the 22 characters `[0-9A-Fa-f]`, so obviously all other code points, Unicode or not, won't match it. (And `\P{ASCII_Hex_Digit}` will match every code point except these 22.)

Getting this message indicates that the outcome of the match arguably should have been the opposite of what actually happened. If you think that is the case, you may wish to make the `non_unicode`

warnings category fatal; if you agree with Perl's decision, you may wish to turn off this category.

See ''Beyond Unicode code points'' in perlunicode(1) for more information.

`%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 whereabouts in the regular expression the problem was discovered. See perlre.

Maximal count of pending signals (%u) exceeded
(F) Perl aborted due to too high a 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'' may clash with future reserved word
(W) This warning may be due to running a perl5(1) 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 in `%s`
(W missing) You called a function with fewer arguments than other arguments you supplied indicated would be needed.

Currently only emitted when a printf-type format required more arguments than were supplied, but might be used in the future for other cases where we can statically determine that arguments to functions are missing, e.g. for the ''pack'' in perlfunc(1) function.

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

Missing braces on \N{}
Missing braces on \N{} in regex; marked by <--HERE in m/%s/
(F) Wrong syntax of character name literal `\N{charname}` within double-quotish context. This can also happen when there is a space (or comment) between the `\N` and the `{` in a regex with the `/x` modifier. This modifier does not change the requirement that the brace immediately follow the `\N`.

Missing braces on \o{}
(F) A `\o` must be followed immediately by a `{` in 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 ’]’ in prototype for `%s : %s`
(W illegalproto) A grouping was started with `[` but never closed with `]`.

Missing name in ‘‘%s sub’’
(F) The 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 or undefined argument to require
(F) You tried to call require with no argument or with an undefined value as an argument. Require expects either a package name or a file-specification as an argument. See ‘‘require’’ in perlfunc.

Missing right brace on \%c{} in regex; marked by <--HERE in m/%s/
(F) Missing right brace in `\x{...}`, `\p{...}`, `\P{...}`, or `\N{...}`.

Missing right brace on \N{}
Missing right brace on \N{} or unescaped left brace after \N
(F) `\N` has two meanings.

The traditional one has it followed by a name enclosed in braces, meaning the character (or sequence of characters) given by that name. Thus `\N{ASTERISK}` is another way of writing `*`, valid in both double-quoted strings and regular expression patterns. In patterns, it doesn’t have the meaning an unescaped `*` does.

Starting in Perl 5.12.0, `\N` also can have an additional meaning (only) in patterns, namely to match a non-newline character. (This is short for `[^\n]`, and like `.` but is not affected by the `/s` regex modifier.)

This can lead to some ambiguities. When `\N` is not followed immediately by a left brace, Perl assumes the `[^\n]` meaning. Also, if the braces form a valid quantifier such as `\N{3}` or `\N{5,}`, Perl assumes that this means to match the given quantity of non-newlines (in these examples, 3; and 5 or more, respectively). In all other case, where there is a `\N{` and a matching `}`, Perl assumes that a character name is desired.

However, if there is no matching `}`, Perl doesn’t know if it was mistakenly omitted, or if `[^\n]{` was desired, and raises this error. If you meant the former, add the right brace; if you meant the latter, escape the brace with a backslash, like so: `\N\{`

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(1) for full details about `-M` and `-m`.

More than one argument to '%s' 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(1) for details.

mprotect for COW string %p %u failed with %d
(S) You compiled perl with **-D**PERL_DEBUG_READONLY_COW (see "Copy on Write" in perlguts), but a shared string buffer could not be made read-only.

mprotect for %p %u failed with %d
(S) You compiled perl with **-D**PERL_DEBUG_READONLY_OPS (see perlhacktips), but an op tree could not be made read-only.

mprotect RW for COW string %p %u failed with %d
(S) You compiled perl with **-D**PERL_DEBUG_READONLY_COW (see "Copy on Write" in perlguts), but a read-only shared string buffer could not be made mutable.

mprotect RW for %p %u failed with %d
(S) You compiled perl with **-D**PERL_DEBUG_READONLY_OPS (see perlhacktips), but a read-only op tree could not be made mutable before freeing the ops.

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.

%s must not be a named sequence in transliteration operator
(F) Transliteration (tr/// and y///) transliterates individual characters. But a named sequence by definition is more than an individual charater, and hence doing this operation on it doesn't make sense.

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

''my'' subroutine %s can't be in a package
(F) Lexically scoped subroutines aren't in a package, so it doesn't make sense to try to declare one with a package qualifier on the front.

''my %s'' used in sort comparison
(W syntax) The package variables $a and $b are used for sort comparisons. You used $a or $b in as an operand to the <=> or cmp operator inside a sort comparison block, 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.

''my'' 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 also provided for this purpose.

NOTE: This warning detects package symbols that have been used only once. This means lexical variables will never trigger this warning. It also means that all of the package variables $c, @c, %c, as well as *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. Symbols beginning with an underscore and symbols using special identifiers (q.v. perldata) are exempt from this warning.

Need exactly 3 octal digits in regex; marked by <--HERE in m/%s/
(F) Within (?[]), all constants interpreted as octal need to be exactly 3 digits long. This helps catch some ambiguities. If your constant is too short, add leading zeros, like

```
(?[ [ \078 ] ]) # Syntax error!
(?[ [ \0078 ] ]) # Works
(?[ [ \007 8 ] ]) # Clearer
```

The maximum number this construct can express is \777. If you need a larger one, you need to use \o{} instead. If you meant two separate things, you need to separate them:

```
(?[ [ \7776 ] ]) # Syntax error!
(?[ [ \o{7776} ] ]) # One meaning
(?[ [ \777 6 ] ]) # Another meaning
(?[ [ \777 \006 ] ]) # Still another
```

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.

Negative repeat count does nothing

(W numeric) You tried to execute the `x` repetition operator fewer than 0 times, which doesn't make sense.

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 whereabouts in the regular expression 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.

\N in a character class must be a named character: \N{...} in regex; marked by <--HERE in m/%s/

(F) The new (as of Perl 5.12) meaning of `\N` as `[^\n]` is not valid in a bracketed character class, for the same reason that `.` in a character class loses its specialness: it matches almost everything, which is probably not what you want.

\N{} in inverted character class or as a range end-point is restricted to one character in regex; marked by <-- HERE in m/%s/

(F) Named Unicode character escapes (`\N{...}`) may return a multi-character sequence. Even though a character class is supposed to match just one character of input, perl will match the whole thing correctly, except when the class is inverted (`[^...]`), or the escape is the beginning or final end point of a range. The mathematically logical behavior for what matches when inverting is very different from what people expect, so we have decided to forbid it. Similarly unclear is what should be generated when the `\N{...}` is used as one of the end points of the range, such as in

```
[\x{41}-\N{ARABIC SEQUENCE YEH WITH HAMZA ABOVE WITH AE}]
```

What is meant here is unclear, as the `\N{...}` escape is a sequence of code points, so this is made an error.

\N{NAME} must be resolved by the lexer in regex; marked by <--HERE in m/%s/

(F) When compiling a regex pattern, an unresolved named character or sequence was encountered. This can happen in any of several ways that bypass the lexer, such as using single-quotish context, or an extra backslash in double-quotish:

```
$re = '\N{SPACE}'; # Wrong!
$re = "\\N{SPACE}"; # Wrong!
/$re/;
```

Instead, use double-quotes with a single backslash:

```
$re = "\N{SPACE}"; # ok
/$re/;
```

The lexer can be bypassed as well by creating the pattern from smaller components:

```
$re = '\N';
/${re}{SPACE}/; # Wrong!
```

It's not a good idea to split a construct in the middle like this, and it doesn't work here. Instead use the solution above.

Finally, the message also can happen under the `/x` regex modifier when the `\N` is separated by spaces from the `{`, in which case, remove the spaces.

```
/\N {SPACE}/x; # Wrong!
/\N{SPACE}/x; # ok
```

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-BREAK SPACE in a charnames alias definition is deprecated

(D deprecated) You defined a character name which contained a no-break space character. Change it to a regular space. Usually these names are defined in the `:alias` import argument to `use charnames`, but they could be defined by a translator installed into `$^H{charnames}`. See "CUSTOM ALIASES" in charnames.

No code specified for -%c

(F) Perl's **-e** and **-E** command-line options require an argument. If you want to run an empty program, pass the empty string as a separate argument or run a program consisting of a single 0 or 1:

```
perl -e ""
perl -e0
perl -e1
```

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(1) 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 directory specified for -I

(F) The **-I** command-line switch requires a directory name as part of the *same* argument. Use **-Ilib**, for instance. **-I lib** won't work.

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 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.

Non-finite repeat count does nothing
(W numeric) You tried to execute the `x` repetition operator `Inf` (or `-Inf`) or `NaN` times, which doesn't make sense.

Non-hex character in regex; marked by <--HERE in m/%s/
(F) In a regular expression, there was a non-hexadecimal character where a hex one was expected, like

```
(?[ [ \xDG ] ])
(?[ [ \x{DEKA} ] ])
```

Non-octal character in regex; marked by <--HERE in m/%s/
(F) In a regular expression, there was a non-octal character where an octal one was expected, like

```
(?[ [ \o{1278} ] ])
```

Non-octal character '%c'. Resolved as ''%s''
(W digit) In parsing an octal numeric constant, a character was unexpectedly encountered that isn't octal. The resulting value is as indicated.

''no'' not allowed in expression
(F) The ''no'' keyword is recognized and executed at compile time, and returns no useful value. See perlmod.

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.

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 rules. 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 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 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 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 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 an unblessed ARRAY reference
(F) You passed a reference to a blessed array to `push`, `shift` or another array function. These only accept unblessed array references or arrays beginning explicitly with `@`.

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.

(?[...]) not valid in locale in regex; marked by <--HERE in m/%s/
(F) `(?[...])` cannot be used within the scope of a `uselocale` or with an `/l` regular expression modifier, as that would require deferring to run-time the calculation of what it should evaluate to, and it is regex compile-time only.

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.

NULL OP IN RUN
(S 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'').

Number with no digits
(F) Perl was looking for a number but found nothing that looked like a number. This happens, for example with `\o{}`, with no number between the braces.

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(1) for more on portability concerns.

Odd name/value argument for subroutine
(F) A subroutine using a slurpy hash parameter in its signature received an odd number of arguments to populate the hash. It requires the arguments to be paired, with the same number of keys as values. The caller of the subroutine is presumably at fault. Inconveniently, this error will be reported at the location of the subroutine, not that of the caller.

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 behavior 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
(D 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
(D 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.

Operand with no preceding operator in regex; marked by <--HERE in m/%s/
(F) You wrote something like

```
(?[ \p{Digit} \p{Thai} ])
```

There are two operands, but no operator giving how you want to combine them.

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 the `fallback` overloading key is specified to be true. See overload.

Operation ''%s'' returns its argument for non-Unicode code point 0x%X
(S non_unicode) You performed an operation requiring Unicode rules on a code point that is not in Unicode, so what it should do is not defined. Perl has chosen to have it do nothing, and warn you.

If the operation shown is ''ToFold'', it means that case-insensitive matching in a regular expression was done on the code point.

If you know what you are doing you can turn off this warning by `no warnings 'non_unicode';`.

Operation ''%s'' returns its argument for UTF-16 surrogate U+%X
(S surrogate) You performed an operation requiring Unicode rules on a Unicode surrogate. Unicode frowns upon the use of surrogates for anything but storing strings in UTF-16, but rules are (reluctantly) defined for the surrogates, and they are to do nothing for this operation. Because the use of surrogates can be dangerous, Perl warns.

If the operation shown is ''ToFold'', it means that case-insensitive matching in a regular expression was done on the code point.

If you know what you are doing you can turn off this warning by `no warnings 'surrogate';`.

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’’’.

Optional parameter lacks default expression
(F) In a subroutine signature, you wrote something like ‘‘$a =’’, making a named optional parameter without a default value. A nameless optional parameter is permitted to have no default value, but a named one must have a specific default. You probably want ‘‘$a = undef’’.

‘‘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.

overload arg ’%s’ is invalid
(W overload) The overload pragma was passed an argument it did not recognize. Did you mistype an operator?

Overloaded dereference did not return a reference
(F) An object with an overloaded dereference operator was dereferenced, but the overloaded operation did not return a reference. See overload.

Overloaded qr did not return a REGEXP
(F) An object with a `qr` overload was used as part of a match, but the overloaded operation didn't return a compiled regexp. See overload.

`%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: child pseudo-process was never scheduled
(P) A child pseudo-process in the ithreads implementation on Windows was not scheduled within the time period allowed and therefore was not able to initialize properly.

panic: ck_grep, type=%u
(P) Failed an internal consistency check trying to compile a grep.

panic: ck_split, type=%u
(P) Failed an internal consistency check trying to compile a split.

panic: corrupt saved stack index `%ld`
(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: 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: `%f`
(P) The library function *frexp()* failed, making printf(''%f'') impossible.

panic: goto, type=%u, ix=%ld
(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: gp_free failed to free glob pointer
(P) The internal routine used to clear a typeglob's entries tried repeatedly, but each time something re-created entries in the glob. Most likely the glob contains an object with a reference back to the glob and a destructor that adds a new object to the glob.

panic: INTERPCASEMOD, `%s`
(P) The lexer got into a bad state at a case modifier.

panic: INTERPCONCAT, %s
(P) The lexer got into a bad state parsing a string with brackets.

panic: kid popen errno read
(F) A forked child returned an incomprehensible message about its errno.

panic: last, type=%u
(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 %u
(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, %s
(P) Something requested a negative number of bytes of malloc.

panic: memory wrap
(P) Something tried to allocate either more memory than possible or a negative amount.

panic: pad_alloc, %p!=%p
(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_free curpad, %p!=%p
(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_free po
(P) A zero scratch pad offset was detected internally. An attempt was made to free a target that had not been allocated to begin with.

panic: pad_reset curpad, %p!=%p
(P) The compiler got confused about which scratch pad it was allocating and freeing temporaries and lexicals from.

panic: pad_sv po
(P) A zero scratch pad offset was detected internally. Most likely an operator needed a target but that target had not been allocated for whatever reason.

panic: pad_swipe curpad, %p!=%p
(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, type=%u
(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, pm=%p, s=%p
(P) Something terrible went wrong in setting up for the split.

panic: realloc, %s
(P) Something requested a negative number of bytes of realloc.

panic: reference miscount on nsv in *sv_replace()* (%d != 1)
(P) The internal *sv_replace()* function was handed a new SV with a reference count other than 1.

panic: restartop in %s
(P) Some internal routine requested a goto (or something like it), and didn't supply the destination.

panic: return, type=%u
(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, %s
(P) *scan_num()* got called on something that wasn't a number.

panic: Sequence (?{...}): no code block found in regex m/%s/
(P) While compiling a pattern that has embedded (?{}) or (??{}) code blocks, perl couldn't locate the code block that should have already been seen and compiled by perl before control passed to the regex compiler.

panic: *strxfrm()* gets absurd - a => %u, ab => %u
(P) The interpreter's sanity check of the C function *strxfrm()* failed. In your current locale the returned transformation of the string ''ab'' is shorter than that of the string ''a'', which makes no sense.

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, midend=%p, bigend=%p
(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: utf16_to_utf8_reversed: odd bytelen
(P) Something tried to call utf16_to_utf8_reversed with an odd (as opposed to even) byte length.

panic: yylex, %s
(P) The lexer got into a bad state while processing a case modifier.

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.

Parsing code internal error (%s)
(F) Parsing code supplied by an extension violated the parser's API in a detectable way.

Passing malformed UTF-8 to ''%s'' is deprecated
(D deprecated, utf8) This message indicates a bug either in the Perl core or in XS code. Such code was trying to find out if a character, allegedly stored internally encoded as UTF-8, was of a given type, such as being punctuation or a digit. But the character was not encoded in legal UTF-8. The%s is replaced by a string that can be used by knowledgeable people to determine what the type being checked against was. If `utf8` warnings are enabled, a further message is raised, giving details of the malformation.

Pattern subroutine nesting without pos change exceeded limit in regex
(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.

`-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()*.)

Perl API version `%s` of `%s` does not match `%s`
(F) The XS module in question was compiled against a different incompatible version of Perl than the one that has loaded the XS module.

Perl folding rules are not up-to-date for 0x%X; please use the
perlbug(1) utility to report; in regex; marked by <--HERE in m/%s/" 4 (S regexp) You used a regular expression with case-insensitive matching, and there is a bug in Perl in which the built-in regular expression folding rules are not accurate. This may lead to incorrect results. Please report this as a bug using the perlbug(1) utility.

PerlIO layer ':win32' is experimental
(S experimental::win32_perlio) The `:win32` PerlIO layer is experimental. If you want to take the risk of using this layer, simply disable this warning:

```
no warnings "experimental::win32_perlio";
```

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 (did you mean `%s`?)--this is only `%s`, stopped
(F) The code you are trying to run has asked for a newer version of Perl than you are running. Perhaps `use 5.10`w as written instead of `use 5.010` or `use v5.10`. Without the leading `v`, the number is interpreted as a decimal, with every three digits after the decimal point representing a part of the version number. So 5.10 is equivalent to v5.100.

Perl `%s` required — this is only `%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''
(X) See ''PERL_SIGNALS'' in perlrun(1) for legal values.

Perls since `%s` too modern — this is `%s`, stopped
(F) The code you are trying to run claims it will not run on the version of Perl you are using because it is too new. Maybe the code needs to be updated, or maybe it is simply wrong and the version check should just be removed.

perl: warning: Non hex character in '$ENV{PERL_HASH_SEED}', seed only partially set
(S) PERL_HASH_SEED should match /^\s*(?:0x)?[0-9a-fA-F]+\s*\z/ but it contained a non hex character. This could mean you are not using the hash seed you think you are.

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, and 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(1) section **LOCALE PROBLEMS**.

perl: warning: strange setting in ’$ENV{PERL_PERTURB_KEYS}’: ’%s’

(S) Perl was run with the environment variable PERL_PERTURB_KEYS defined but containing an unexpected value. The legal values of this setting are as follows.

```
Numeric | String | Result
--------+---------------+-----------------------------------------
0 | NO | Disables key traversal randomization
1 | RANDOM | Enables full key traversal randomization
2 | DETERMINISTIC | Enables repeatable key traversal
| | randomization
```

Both numeric and string values are accepted, but note that string values are case sensitive. The default for this setting is ‘‘RANDOM’’ or 1.

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 ‘‘*’’.

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 whereabouts in the regular expression 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 [%c %c] belongs inside character classes%s in regex; marked by <--HERE in m/%s/

(W regexp) Perl thinks that you intended to write a POSIX character class, but didn’t use enough brackets. These POSIX class constructs [: :], [= =], and [. .] go *inside* character classes, the [] are part of the construct, for example: `qr/[012[:alpha:]345]/`. What the regular expression pattern compiled to is probably not what you were intending. For example, `qr/[:alpha:]/` compiles to a regular bracketed character class consisting of the four characters `":"`, `"a"`, `"l"`, `"h"`, and `"p"`. To specify the POSIX class, it should have been written `qr/[[:alpha:]]/`.

Note that [= =] and [. .] are not currently implemented; they are simply placeholders for future extensions and will cause fatal errors. The <--HERE shows whereabouts in the regular expression the problem was discovered. See perlre.

If the specification of the class was not completely valid, the message indicates that.

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 whereabouts in the regular expression 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 whereabouts in the regular expression 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 issue with control flow operator

(W syntax) There is a possible problem with the mixing of a control flow operator (e.g. `return`) and a low-precedence operator like `or`. Consider:

```
sub { return $a or $b; }
```

This is parsed as:

```
sub { (return $a) or $b; }
```

Which is effectively just:

```
sub { return $a; }
```

Either use parentheses or the high-precedence variant of the operator.

Note this may be also triggered for constructs like:

```
sub { 1 if die; }
```

Possible precedence problem on bitwise %s 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 $\ in regex

(W ambiguous) You said something like `m/$\/` in a regex. The regex `m/foo$\s+bar/m` translates to: match the word 'foo', the output record separator (see ''$\'' in perlvar) and the letter 's' (one time or more) followed by the word 'bar'.

If this is what you intended then you can silence the warning by using `m/${\}/` (for example: `m/foo${\}s+bar/`).

If instead you intended to match the word 'foo' at the end of the line followed by whitespace and the word 'bar' on the next line then you can use `m/$(?)\/` (for example: `m/foo$(?)\s+bar/`).

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.

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 after '%c' for %s : %s

(W illegalproto) A character follows % or @ in a prototype. This is useless, since % and @ gobble the rest of the subroutine arguments.

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.

Prototype '%s' overridden by attribute 'prototype(%s)' in %s
(W prototype) A prototype was declared in both the parentheses after the sub name and via the prototype attribute. The prototype in parentheses is useless, since it will be replaced by the prototype from the attribute before it's ever used.

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 whereabouts in the regular expression 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 whereabouts in the regular expression the problem was discovered. See perlre.

Quantifier {n,m} with n > m can't match in regex
Quantifier {n,m} with n > m can't match in regex; marked by <--HERE in m/%s/
(W regexp) Minima should be less than or equal to maxima. If you really want your regexp to match something 0 times, just put {0}.

Quantifier unexpected on zero-length expression in regex 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}/`.

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.

Ranges of ASCII printables should be some subset of "0-9", "A-Z", or "a-z" in regex; marked by <--HERE in m/%s/
(W regexp) (only under `usere'strict'` or within `(?[...])`)

Stricter rules help to find typos and other errors. Perhaps you didn't even intend a range here, if the `"-"` was meant to be some other character, or should have been escaped (like `"\-"`). If you did intend a range, the one that was used is not portable between ASCII and EBCDIC platforms, and doesn't have an obvious meaning to a casual reader.

```
[3-7] # OK; Obvious and portable
[d-g] # OK; Obvious and portable
[A-Y] # OK; Obvious and portable
[A-z] # WRONG; Not portable; not clear what is meant
[a-Z] # WRONG; Not portable; not clear what is meant
[%-.] # WRONG; Not portable; not clear what is meant
[\x41-Z] # WRONG; Not portable; not obvious to non-geek
```

(You can force portability by specifying a Unicode range, which means that the endpoints are specified by `\N{...}`, but the meaning may still not be obvious.) The stricter rules require that ranges that start or stop with an ASCII character that is not a control have all their endpoints be the literal character, and not some escape sequence (like `"\x41"`), and the ranges must be all digits, or all uppercase letters, or all lowercase letters.

Ranges of digits should be from the same group in regex; marked by <--HERE in m/%s/
(W regexp) (only under `usere'strict'` or within `(?[...])`)

Stricter rules help to find typos and other errors. You included a range, and at least one of the end points is a decimal digit. Under the stricter rules, when this happens, both end points should be digits in the same group of 10 consecutive digits.

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: %x
(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
(S 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 call to Perl_load_module in PerlIO_find_layer
(P) It is currently not permitted to load modules when creating a filehandle inside an %INC hook. This can happen with `open my $fh, '<', \$scalar`, which implicitly loads PerlIO::scalar. Try loading PerlIO::scalar explicitly first.

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.

Redundant argument in %s
(W redundant) You called a function with more arguments than other arguments you supplied indicated would be needed. Currently only emitted when a printf-type format required fewer arguments than were supplied, but might be used in the future for e.g. ''pack'' in perlfunc.

refcnt_dec: fd %d%s
refcnt: fd %d%s
refcnt_inc: fd %d%s
(P) Perl's I/O implementation failed an internal consistency check. If you see this message, something is very wrong.

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 to invalid group 0 in regex; marked by <--HERE in m/%s/
(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). 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 ordinal 7 inserted into the regular expression, prepend zeroes to make it three digits long: `\007`

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

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

Regexp modifier "/%c" may appear a maximum of twice
Regexp modifier "%c" may appear a maximum of twice in regex; marked by <--HERE in m/%s/
(F) The regular expression pattern had too many occurrences of the specified modifier. Remove the extraneous ones.

Regexp modifier "%c" may not appear after the "-" in regex; marked by <-- HERE in m/%s/
(F) Turning off the given modifier has the side effect of turning on another one. Perl currently doesn't allow this. Reword the regular expression to use the modifier you want to turn on (and place it before the minus), instead of the one you want to turn off.

Regexp modifier "/%c" may not appear twice
Regexp modifier "%c" may not appear twice in regex; marked by <-- HERE in m/%s/
(F) The regular expression pattern had too many occurrences of the specified modifier. Remove the extraneous ones.

Regexp modifiers "/%c" and "/%c" are mutually exclusive
Regexp modifiers "%c" and "%c" are mutually exclusive in regex; marked by <--HERE in m/%s/
(F) The regular expression pattern had more than one of these mutually exclusive modifiers. Retain only the modifier that is supposed to be there.

Regexp out of space in regex m/%s/
(P) A "can't happen" error, because *safemalloc()* should have caught it earlier.

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

Replacement list is longer than search list
(W misc) You have used a replacement list that is longer than the search list. So the additional elements in the replacement list are meaningless.

'%s' resolved to '\o{%s}%d'
(W misc, regexp) You wrote something like `\08`, or `\179` in a double-quotish string. All but the last digit is treated as a single character, specified in octal. The last digit is the next character in the string. To tell Perl that this is indeed what you want, you can use the `\o{ }` syntax, or use exactly three digits to specify the octal for the character.

Reversed `%s`= operator
(W syntax) You wrote your assignment operator backwards. The = must always come 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.

Scalars leaked: %d

(S internal) 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.

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 v ariable `$m` may cause this error.

Note that since Perl 5.10.0 a // can also be the *defined-or* construct, not just the empty search pattern. Therefore code written in Perl 5.10.0 or later that uses the // as the *defined-or* can be misparsed by pre-5.10.0 Perls as a non-terminated search pattern.

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.

%*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.

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 "\c{" invalid
(F) These three characters may not appear in sequence in a double-quotish context. This message is raised only on non-ASCII platforms (a different error message is output on ASCII ones). If you were intending to specify a control character with this sequence, you'll have to use a different way to specify it.

Sequence (? incomplete in regex; marked by <--HERE in m/%s/
(F) A regular expression ended with an incomplete extension (?. The <--HERE shows whereabouts in the regular expression the problem was discovered. See perlre.

Sequence (?%c...) 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 whereabouts in the regular expression 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 whereabouts in the regular expression the problem was discovered. This may happen when using the `(?^...)` construct to tell Perl to use the default regular expression modifiers, and you redundantly specify a default modifier. For other causes, see perlre.

Sequence (?#... not terminated in regex m/%s/
(F) A regular expression comment must be terminated by a closing parenthesis. Embedded parentheses aren't allowed. See perlre.

Sequence (?&... not terminated in regex; marked by <--HERE in m/%s/
(F) A named reference of the form `(?&...)` was missing the final closing parenthesis after the name. The <--HERE shows whereabouts in the regular expression the problem was discovered.

Sequence (?%c... not terminated in regex; marked by <--HERE in m/%s/
(F) A named group of the form `(?'...')` or `(?<...>)` was missing the final closing quote or angle bracket. The <--HERE shows whereabouts in the regular expression the problem was discovered.

Sequence (?(%c... not terminated in regex; marked by <--HERE in m/%s/
(F) A named reference of the form `(?('...')...)` or `(?(<...>)...)` was missing the final closing quote or angle bracket after the name. The <--HERE shows whereabouts in the regular expression the problem was discovered.

Sequence (?... not terminated in regex; marked by <--HERE in m/%s/
(F) There was no matching closing parenthesis for the '('. The <--HERE shows whereabouts in the regular expression the problem was discovered.

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 with ')'
(F) The end of the perl code contained within the {...} must be followed immediately by a ')'.

Sequence (?P>... not terminated in regex; marked by <--HERE in m/%s/
(F) A named reference of the form `(?P>...)` was missing the final closing parenthesis after the name. The <--HERE shows whereabouts in the regular expression the problem was discovered.

Sequence (?P<... not terminated in regex; marked by <--HERE in m/%s/
(F) A named group of the form `(?P<...>')` was missing the final closing angle bracket. The <--HERE shows whereabouts in the regular expression the problem was discovered.

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

(F) A named reference of the form `(?P=...)` was missing the final closing parenthesis after the name. The <--HERE shows whereabouts in the regular expression the problem was discovered.

Sequence (?R) not terminated in regex m/%s/

(F) An `(?R)` or `(?0)` sequence in a regular expression was missing the final parenthesis.

Server error (a.k.a. ‘‘500 Server error’’)

(A) 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.

Setting ${^ENCODING} is deprecated

(D deprecated) You assigned a non-`undef` value to `${^ENCODING}`. This is deprecated; see `"${^ENCODING}" in perlvar(1)` for details.

Setting $/ to a reference to %s as a form of slurp is deprecated, treating as undef

(D deprecated) You assigned a reference to a scalar to `$/` where the referenced item is not a positive integer. In older perls this **appeared** to work the same as setting it to `undef` but was in fact internally different, less efficient and with very bad luck could have resulted in your file being split by a stringified form of the reference.

In Perl 5.20.0 this was changed so that it would be **exactly** the same as setting `$/` to undef, with the exception that this warning would be thrown.

You are recommended to change your code to set `$/` to `undef` explicitly if you wish to slurp the file. In future versions of Perl assigning a reference to will throw a fatal error.

Setting $/ to `%s` reference is forbidden
(F) You tried to assign a reference to a non integer to `$/`. In older Perls this would have behaved similarly to setting it to a reference to a positive integer, where the integer was the address of the reference. As of Perl 5.20.0 this is a fatal error, to allow future versions of Perl to use non-integer refs for more interesting purposes.

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.

/%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?

Slab leaked from cv `%p`
(S) If you see this message, then something is seriously wrong with the internal bookkeeping of op trees. An op tree needed to be freed after a compilation error, but could not be found, so it was leaked instead.

sleep(%u) too large
(W overflow) You called `sleep` with a number that was larger than it can reliably handle and `sleep` probably slept for less time than requested.

Slurpy parameter not last
(F) In a subroutine signature, you put something after a slurpy (array or hash) parameter. The slurpy parameter takes all the available arguments, so there can't be any left to fill later parameters.

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.

Smartmatch is experimental
(S experimental::smartmatch) This warning is emitted if you use the smartmatch (~~) operator. This is currently an experimental feature, and its details are subject to change in future releases of Perl. Particularly, its current behavior is noticed for being unnecessarily complex and unintuitive, and is very likely to be overhauled.

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.

Source filters apply only to byte streams
(F) You tried to activate a source filter (usually by loading a source filter module) within a string passed to `eval`. This is not permitted under the `unicode_eval` feature. Consider using `evalbytes` instead. See feature.

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.

‘‘state’’ subroutine `%s` can’t be in a package
(F) Lexically scoped subroutines aren’t in a package, so it doesn’t make sense to try to declare one with a package qualifier on the front.

‘‘state `%s`’’ used in sort comparison
(W syntax) The package variables `$a` and `$b` are used for sort comparisons. You used `$a` or `$b` in as an operand to the `<=>` or `cmp` operator inside a sort comparison block, 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.

‘‘state’’ 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.

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.

Strings with code points over 0xFF may not be mapped into in-memory file handles
(W utf8) You tried to open a reference to a scalar for read or append where the scalar contained code points over 0xFF. In-memory files model on-disk files and can only contain bytes.

Stub found while resolving method ‘‘%s’’ overloading ‘‘%s’’ in package ‘‘%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’’ is not available
(W closure) During compilation, an inner named subroutine or eval is attempting to capture an outer lexical subroutine that is not currently available. This can happen for one of two reasons. First, the lexical subroutine 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 sub a {...} sub f { \&a } }
```

At the time that f is created, it can’t capture the current ‘‘a’’ sub, 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 sub a {...} eval 'sub f { \&a }' }->();
```

The second situation is caused by an eval accessing a lexical subroutine that has gone out of scope, for example,

```
sub f {
my sub 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.

‘‘%s’’ subroutine &%s masks earlier declaration in same %s
(W misc) A ‘‘my’’ or ‘‘state’’ subroutine 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 subroutine will still exist until the end of the scope or until all closure references to it are destroyed.

Subroutine %s redefined
(W redefine) You redefined a subroutine. To suppress this warning, say

```
{
no warnings 'redefine';
eval "sub name { ... }";
}
```

Subroutine ‘‘%s’’ will not stay shared
(W closure) An inner (nested) *named* subroutine is referencing a ‘‘my’’ subroutine defined in an outer named subroutine.

When the inner subroutine is called, it will see the value of the outer subroutine’s lexical subroutine 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 lexical subroutine. In other words, it will no longer be shared. This will especially make a difference if the lexical subroutines accesses lexical variables declared in its surrounding scope.

This problem can usually be solved by making the inner subroutine anonymous, using the `sub {}` syntax. When inner anonymous subs that reference lexical subroutines in outer subroutines are created, they are automatically rebound to the current values of such lexical subs.

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 v ariable `$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 v ariable `$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).

sv_upgrade from type %d down to type %d
(P) Perl tried to force the upgrade of an SV to a type which was actually inferior to its current type.

SWASHNEW didn’t return an HV ref
(P) Something went wrong internally when Perl was trying to look up Unicode characters.

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 whereabouts in the regular expression the problem was discovered. See perlre.

Switch condition not recognized 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 must be one of the following:

```
(1) (2) ... true if 1st, 2nd, etc., capture matched
(<NAME>) ('NAME') true if named capture matched
(?=...) (?<=...) true if subpattern matches
(?!...) (?<!...) true if subpattern fails to match
(?{ CODE }) true if code returns a true value
(R) true if evaluating inside recursion
(R1) (R2) ... true if directly inside capture group 1, 2, etc.
(R&NAME) true if directly inside named capture
(DEFINE) always false; for defining named subpatterns
```

The <--HERE shows whereabouts in the regular expression the problem was discovered. See perlre.

Switch (?(condition)... not terminated in regex; marked by <--HERE in m/%s/

(F) You omitted to close a (?(condition)...) block somewhere in the pattern. Add a closing parenthesis in the appropriate position. 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.

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 20questions.

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(1) script through a perl4 interpreter, especially if the next 2 tokens are ``use strict'' or ``my `$var`'' or ``our `$var`''.

Syntax error in (?[...]) in regex; marked by <-- HERE in m/%s/

(F) Perl could not figure out what you meant inside this construct; this notifies you that it is giving up trying.

%s syntax OK

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

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.

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.

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.

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(1) and arybase.

The bitwise feature is experimental
(S experimental::bitwise) This warning is emitted if you use bitwise operators (& | ^ ~ &. |. ^. ~.) with the ‘‘bitwise’’ feature enabled. Simply suppress the warning if you want to use the feature, but know that in doing so you are taking the risk of using an experimental feature which may change or be removed in a future Perl version:

```
no warnings "experimental::bitwise";
use feature "bitwise";
$x |.= $y;
```

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
(F) The function indicated isn’t implemented on this architecture, according to the probings of Configure.

The lexical_subs feature is experimental
(S experimental::lexical_subs) This warning is emitted if you declare a sub with my or state. Simply suppress the warning if you want to use the feature, but know that in doing so you are taking the risk of using an experimental feature which may change or be removed in a future Perl version:

```
no warnings "experimental::lexical_subs";
use feature "lexical_subs";
my sub foo { ... }
```

The regex_sets feature is experimental

(S experimental::regex_sets) This warning is emitted if you use the syntax `(?[])` in a regular expression. The details of this feature are subject to change. if you want to use it, but know that in doing so you are taking the risk of using an experimental feature which may change in a future Perl version, you can do this to silence the warning:

```
no warnings "experimental::regex_sets";
```

The signatures feature is experimental

(S experimental::signatures) This warning is emitted if you unwrap a subroutine's arguments using a signature. Simply suppress the warning if you want to use the feature, but know that in doing so you are taking the risk of using an experimental feature which may change or be removed in a future Perl version:

```
no warnings "experimental::signatures";
use feature "signatures";
sub foo ($left, $right) { ... }
```

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.

This Perl has not been built with support for randomized hash key traversal but something called *Perl_hv_rand_set()*.

(F) Something has attempted to use an internal API call which depends on Perl being compiled with the default support for randomized hash key traversal, but this Perl has been compiled without it. You should report this warning to the relevant upstream party, or recompile perl with default options.

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 (or 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 **-%c** option is a part of Perl's first argument: e.g. change `perl -n -%c` to `perl -%c -n`.

If the Perl script is being executed as `perl scriptname`, then the **-%c** option must appear on the command line: `perl -%c 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 few arguments for subroutine
(F) A subroutine using a signature received fewer arguments than required by the signature. The caller of the subroutine is presumably at fault. Inconveniently, this error will be reported at the location of the subroutine, not that of the caller.

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 arguments for subroutine
(F) A subroutine using a signature received more arguments than required by the signature. The caller of the subroutine is presumably at fault. Inconveniently, this error will be reported at the location of the subroutine, not that of the caller.

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 &CORE::%s must be `%s`
(F) The subroutine in question in the CORE package requires its argument to be a hard reference to data of the specified type. Overloading is ignored, so a reference to an object that is not the specified

type, but nonetheless has overloading to handle it, will still not be accepted.

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).

Unbalanced context: %d more PUSHes than POPs
(S internal) The exit code detected an internal inconsistency in how many execution contexts were entered and left.

Unbalanced saves: %d more saves than restores
(S internal) The exit code detected an internal inconsistency in how many values were temporarily localized.

Unbalanced scopes: %d more ENTERs than LEAVEs
(S internal) The exit code detected an internal inconsistency in how many blocks were entered and left.

Unbalanced string table refcount: (%d) for ''%s''
(S internal) On exit, Perl found some strings remaining in the shared string table used for copy on write and for hash keys. The entries should have been freed, so this indicates a bug somewhere.

Unbalanced tmps: %d more allocs than frees
(S 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.

Unescaped left brace in regex is deprecated, passed through in regex; marked by <--HERE in m/%s/
(D deprecated, regexp) You used a literal `"{"` character in a regular expression pattern. You should change to use `"\{"` instead, because a future version of Perl (tentatively v5.26) will consider this to be a syntax error. If the pattern delimiters are also braces, any matching right brace (`"}"`) should also be escaped to avoid confusing the parser, for example,

```
qr{abc\{def\}ghi}
```

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.

Unexpected ']' with no following ')' in (?[... in regex; marked by <-- HERE in m/%s/

(F) While parsing an extended character class a ']' character was encountered at a point in the definition where the only legal use of ']' is to close the character class definition as part of a '])', you may have forgotten the close paren, or otherwise confused the parser.

Expecting close paren for nested extended charclass in regex; marked by <-- HERE in m/%s/

(F) While parsing a nested extended character class like:

```
(?[ ... (?flags:(?[ ... ])) ... ])
    ^
```

we expected to see a close paren ')' (marked by ^) but did not.

Expecting close paren for wrapper for nested extended charclass in regex; marked by <-- HERE in m/%s/

(F) While parsing a nested extended character class like:

```
(?[ ... (?flags:(?[ ... ])) ... ])
    ^
```

we expected to see a close paren ')' (marked by ^) but did not.

Unexpected binary operator '%c' with no preceding operand in regex; marked by <--HERE in m/%s/

(F) You had something like this:

```
(?[ | \p{Digit} ])
```

where the "|" is a binary operator with an operand on the right, but no operand on the left.

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

(F) You had something like this:

```
(?[ z ])
```

Within (?[ ]), no literal characters are allowed unless they are within an inner pair of square brackets, like

```
(?[ [ z ] ])
```

Another possibility is that you forgot a backslash. Perl isn't smart enough to figure out what you really meant.

Unexpected constant lvalue entersub entry via type/targ %d:%d

(P) When compiling a subroutine call in lvalue context, Perl failed an internal consistency check. It encountered a malformed op tree.

Unexpected exit %u

(S) *exit()* was called or the script otherwise finished gracefully when PERL_EXIT_WARN was set in PL_exit_flags.

Unexpected exit failure %d

(S) An uncaught *die()* was called when PERL_EXIT_WARN was set in PL_exit_flags.

Unexpected ')' in regex; marked by <--HERE in m/%s/

(F) You had something like this:

```
(?[ ( \p{Digit} + ) ])
```

The ")" is out-of-place. Something apparently was supposed to be combined with the digits, or the "+" shouldn't be there, or something like that. Perl can't figure out what was intended.

Unexpected '(' with no preceding operator in regex; marked by <--HERE in m/%s/
(F) You had something like this:

```
(?[ \p{Digit} ( \p{Lao} + \p{Thai} ) ])
```

There should be an operator before the "(", as there's no indication as to how the digits are to be combined with the characters in the Lao and Thai scripts.

Unicode non-character U+%X is not recommended for open interchange
(S nonchar) Certain codepoints, such as U+FFFE and U+FFFF, are defined by the Unicode standard to be non-characters. Those are legal codepoints, but are reserved for internal use; so, applications shouldn't attempt to exchange them. An application may not be expecting any of these characters at all, and receiving them may lead to bugs. If you know what you are doing you can turn off this warning by `no warnings 'nonchar';`.

This is not really a ''severe'' error, but it is supposed to be raised by default even if warnings are not enabled, and currently the only way to do that in Perl is to mark it as serious.

Unicode surrogate U+%X is illegal in UTF-8
(S surrogate) You had a UTF-16 surrogate in a context where they are not considered acceptable. These code points, between U+D800 and U+DFFF (inclusive), are used by Unicode only for UTF-16. However, Perl internally allows all unsigned integer code points (up to the size limit available on your platform), including surrogates. But these can cause problems when being input or output, which is likely where this message came from. If you really really know what you are doing you can turn off this warning by `no warnings 'surrogate';`.

Unknown charname '' is deprecated
(D deprecated) You had a `\N{}` with nothing between the braces. This usage is deprecated, and will be made a syntax error in a future Perl version.

Unknown charname '%s'
(F) The name you used inside `\N{}` is unknown to Perl. Check the spelling. You can say `use charnames ":loose"`to not ha ve to be so precise about spaces, hyphens, and capitalization on standard Unicode names. (Any custom aliases that have been created must be specified exactly, regardless of whether `:loose` is used or not.) This error may also happen if the `\N{}` is not in the scope of the corresponding `usecharnames`.

Unknown error
(P) Perl was about to print an error message in `$@`, but the `$@` variable did not exist, even after an attempt to create it.

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 regex modifier ''%s''
(F) Alphanumerics immediately following the closing delimiter of a regular expression pattern are interpreted by Perl as modifier flags for the regex. One of the ones you specified is invalid. One way this can happen is if you didn't put in white space between the end of the regex and a following alphanumeric operator:

```
if ($a =~ /foo/and $bar == 3) { ... }
```

The "a" is a valid modifier flag, but the "n" is not, and raises this error. Likely what was meant instead was:

```
if ($a =~ /foo/ and $bar == 3) { ... }
```

Unknown ‘‘re’’ subpragma ’%s’ (known ones are: %s)
(W) You tried to use an unknown subpragma of the ‘‘re’’ pragma.

Unknown switch condition (?(...)) 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 must be one of the following:

```
(1) (2) ... true if 1st, 2nd, etc., capture matched
(<NAME>) ('NAME') true if named capture matched
(?=...) (?<=...) true if subpattern matches
(?!...) (?<!...) true if subpattern fails to match
(?{ CODE }) true if code returns a true value
(R) true if evaluating inside recursion
(R1) (R2) ... true if directly inside capture group 1, 2, etc.
(R&NAME) true if directly inside named capture
(DEFINE) always false; for defining named subpatterns
```

The <--HERE shows whereabouts in the regular expression the problem was discovered. See perlre.

Unknown Unicode option letter ’%c’
(F) You specified an unknown Unicode option. See perlrun(1) documentation of the -C switch for the list of known options.

Unknown Unicode option value %d
(F) You specified an unknown Unicode option. See perlrun(1) documentation of the -C switch for the list of known options.

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(1) for details on legal verb patterns.

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 loaded this module 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 whereabouts in the regular expression the problem was discovered. See perlre.

Unmatched (in regex; marked by <--HERE in m/%s/
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 whereabouts in the regular expression 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`; marked by <--HERE after `%s` near column `%d`
(F) The Perl parser has no idea what to do with the specified character in your Perl script (or eval) near 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 in regex; marked by <--HERE in m/%s/
(F) You used a backslash-character combination which is not recognized by Perl inside character classes. This is a fatal error when the character class is used within `(?[ ])`.

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, but this may change in a future version of Perl. The <--HERE shows whereabouts in the regular expression 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, but this may change in a future version of Perl.

Unrecognized escape \%s 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(s) were understood literally, but this may change in a future version of Perl. The <--HERE shows whereabouts in the regular expression 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 delimiter for here document
(F) This message occurs when a here document label has an initial quotation mark but the final quotation mark is missing. Perhaps you wrote:

```
<<"foo
```

instead of:

```
<<"foo"
```

Unterminated \g... pattern in regex; marked by <--HERE in m/%s/
Unterminated \g{...} pattern in regex; marked by <--HERE in m/%s/
(F) In a regular expression, you had a `\g` that wasn't followed by a proper group reference. In the case of `\g{`, the closing brace is missing; otherwise the `\g` must be followed by an integer. 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".

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 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.

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.

$[used in `%s` (did you mean $] ?)
(W syntax) You used `$[` in a comparison, such as:

```
if ($[ > 5.006) {
...
}
```

You probably meant to use `$]` instead. `$[` is the base for indexing arrays. `$]` is the Perl version number in decimal.

Use ‘‘%s’’ instead of ‘‘%s’’
(F) The second listed construct is no longer legal. Use the first one instead.

Useless assignment to a temporary
(W misc) You assigned to an lvalue subroutine, but what the subroutine returned was a temporary scalar about to be discarded, so the assignment had no effect.

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 whereabouts in the regular expression 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 whereabouts in the regular expression the problem was discovered. See perlre.

Useless use of attribute ‘‘const’’
(W misc) The `const` attribute has no effect except on anonymous closure prototypes. You applied it to a subroutine via attributes.pm. This is only useful inside an attribute handler for an anonymous subroutine.

Useless use of /d modifier in transliteration operator
(W misc) You have used the /d modifier where the searchlist has the same length as the replacelist. See perlop(1) for more information about the /d modifier.

Useless use of \E
(W misc) You have a \E in a double-quotish string without a `\U`, `\L` or `\Q` preceding it.

Useless use of greediness modifier ’%c’ in regex; marked by <--HERE in m/%s/
(W regexp) You specified something like these:

```
qr/a{3}?/
qr/b{1,1}+/
```

The `"?"` and `"+"` don’t have any effect, as they modify whether to match more or fewer when there is a choice, and by specifying to match exactly a given numer, there is no room left for a choice.

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(1) 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 (?-p) in regex; marked by <--HERE in m/%s/
(W regexp) The `p` modifier cannot be turned off once set. Trying to do so is futile.

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 assignment to $[is deprecated
(D deprecated) The `$[` variable (index of the first element in an array) is deprecated. See ''$['' in perlvar.

Use of bare << to mean <<"" is deprecated
(D deprecated) 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 /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 code point 0x%s is deprecated; the permissible max is 0x%s
(D deprecated) You used a code point that will not be allowed in a future perl version, because it is too large. Unicode only allows code points up to 0x10FFFF, but Perl allows much larger ones. However, the largest possible ones break the perl interpreter in some constructs, including causing it to hang in a few cases. The known problem areas are in `tr///`, regular expression pattern matching using quantifiers, and as the upper limits in loops.

If your code is to run on various platforms, keep in mind that the upper limit depends on the platform. It is much larger on 64-bit word sizes than 32-bit ones.

Use of comma-less variable list is deprecated
(D deprecated) The values you give to a format should be separated by commas, not just aligned on a line.

Use of *each()* on hash after insertion without resetting hash iterator results in undefined behavior
(S internal) The behavior of `each()` after insertion is undefined; it may skip items, or visit items more than once. Consider using `keys()` instead of `each()`.

Use of := for an empty attribute list is not allowed
(F) The construction `my $x := 42` used to parse as equivalent to `my $x : = 42` (applying an empty attribute list to `$x`). This construct was deprecated in 5.12.0, and has now been made a syntax error, so `:=` can be reclaimed as a new operator in the future.

If you need an empty attribute list, for example in a code generator, add a space before the `=`.

Use of `%s` for non-UTF-8 locale is wrong. Assuming a UTF-8 locale
(W locale) You are matching a regular expression using locale rules, and the specified construct was encountered. This construct is only valid for UTF-8 locales, which the current locale isn't. This doesn't make sense. Perl will continue, assuming a Unicode (UTF-8) locale, but the results are likely to be wrong.

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 ''goto'' to jump into a construct is deprecated
(D deprecated) Using `goto` to jump from an outer scope into an inner scope is deprecated and should be avoided.

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 literal control characters in variable names is deprecated

Use of literal non-graphic characters in variable names is deprecated

(D deprecated) Using literal non-graphic (including control) characters in the source to refer to the ^FOO variables, like `$^X` and `${^GLOBAL_PHASE}` is now deprecated. (We use `^X` and `^G` here for legibility. They actually represent the non-printable control characters, code points 0x18 and 0x07, respectively; `^A` would mean the control character whose code point is 0x01.) This only affects code like `$\cT`, where `\cT` is a control in the source code; `${"\cT"}` and `$^T` remain valid. Things that are non-controls and also not graphic are NO-BREAK SPACE and SOFT HYPHEN, which were previously only allowed for historical reasons.

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 %s on a handle without * is deprecated

(D deprecated) You used `tie`, `tied` or `untie` on a scalar but that scalar happens to hold a typeglob, which means its filehandle will be tied. If you mean to tie a handle, use an explicit * as in `tie *$handle`.

This was a long-standing bug that was removed in Perl 5.16, as there was no way to tie the scalar itself when it held a typeglob, and no way to untie a scalar that had had a typeglob assigned to it. If you see this message, you must be using an older version.

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, however, because you can overload the numification and stringification operators and then you presumably know what you are doing.

Use of state $_ is experimental

(S experimental::lexical_topic) Lexical $_ is an experimental feature and its behavior may change or even be removed in any future release of perl. See the explanation under "$_" in perlvar.

Use of strings with code points over 0xFF as arguments to %s operator is deprecated

(D deprecated) You tried to use one of the string bitwise operators (`&` or `|` or `^` or `~`) on a string containing a code point over 0xFF. The string bitwise operators treat their operands as strings of bytes, and values beyond 0xFF are nonsensical in this context.

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.

‘‘use re ’strict’’’ is experimental

(S experimental::re_strict) The things that are different when a regular expression pattern is compiled under `'strict'` are subject to change in future Perl releases in incompatible ways. This means that a pattern that compiles today may not in a future Perl release. This warning is to alert you to that risk.

Use \x{...} for more than two hex characters in regex; marked by <--HERE in m/%s/

(F) In a regular expression, you said something like

```
(?[ [ \xBEEF ] ])
```

Perl isn’t sure if you meant this

```
(?[ [ \x{BEEF} ] ])
```

or if you meant this

```
(?[ [ \x{BE} E F ] ])
```

You need to add either braces or blanks to disambiguate.

Using just the first character returned by \N{} in character class in regex; marked by <--HERE in m/%s/

(W regexp) Named Unicode character escapes (`\N{...}`) may return a multi-character sequence. Even though a character class is supposed to match just one character of input, perl will match the whole thing correctly, except when the class is inverted (`[^...]`), or the escape is the beginning or final end point of a range. For these, what should happen isn’t clear at all. In these circumstances, Perl discards all but the first character of the returned sequence, which is not likely what you want.

Using /u for ’%s’ instead of /%s in regex; marked by <--HERE in m/%s/

(W regexp) You used a Unicode boundary (`\b{...}` or `\B{...}`) in a portion of a regular expression where the character set modifiers `/a` or `/aa` are in effect. These two modifiers indicate an ASCII interpretation, and this doesn’t make sense for a Unicode defintion. The generated regular expression will compile so that the boundary uses all of Unicode. No other portion of the regular expression is affected.

Using !~ with %s doesn’t make sense

(F) Using the `!~` operator with `s///r`, `tr///r` or `y///r` is currently reserved for future use, as the exact behavior has not been decided. (Simply returning the boolean opposite of the modified string is usually not particularly useful.)

UTF-16 surrogate U+%X

(S surrogate) You had a UTF-16 surrogate in a context where they are not considered acceptable. These code points, between U+D800 and U+DFFF (inclusive), are used by Unicode only for UTF-16. However, Perl internally allows all unsigned integer code points (up to the size limit available on your platform), including surrogates. But these can cause problems when being input or output, which is likely where this message came from. If you really really know what you are doing you can turn off this warning by `no warnings 'surrogate';`.

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

(S misc) With ‘‘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 regex m/%s/

(F) Lookbehind is allowed only for subexpressions whose length is fixed and known at compile time. For positive lookbehind, you can use the `\K` regex construct as a way to get the equivalent functionality. See ‘‘(?<=pattern) \K’’ in perlre.

There are non-obvious Unicode rules under `/i` that can match variably, but which you might not think could. For example, the substring`"ss"` can match the single character LA TIN SMALL LETTER SHARP S. There are other sequences of ASCII characters that can match single ligature characters, such as LATIN SMALL LIGATURE FFI matching `qr/ffi/i`. Starting in Perl v5.16, if you only care about ASCII matches, adding the `/aa` modifier to the regex will exclude all these non-obvious matches, thus getting rid of this message. You can also say `usereqw(/aa)` to apply `/aa` to all regular expressions compiled within its scope. See re.

‘‘%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 references 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.

vector argument not supported with alpha versions
(S printf) The `%vd` (s)printf format does not support version objects with alpha parts.

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: unable to close filehandle properly: `%s`
Warning: unable to close filehandle `%s` properly: `%s`
(S io) There were errors during the implicit *close()* done on a filehandle when its reference count reached zero while it was still open, e.g.:

```
{
open my $fh, '>', $file or die "open: '$file': $!\n";
print $fh $data or die "print: $!";
} # implicit close here
```

Because various errors may only be detected by *close()* (e.g. buffering could allow the `print` in this example to return true even when the disk is full), it is dangerous to ignore its result. So when it happens implicitly, perl will signal errors by warning.

Prior to version 5.22.0, perl ignored such errors, so the common idiom shown above was liable to cause **silent data loss**.

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.

when is experimental

(S experimental::smartmatch) `when` depends on smartmatch, which is experimental. Additionally, it has several special cases that may not be immediately obvious, and their behavior may change or even be removed in any future release of perl. See the explanation under ‘‘Experimental Details on given and when’’ in perlsyn.

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.

Wide character (U+%X) in `%s`

(W locale) While in a single-byte locale (*i.e.*, a non-UTF-8 one), a multi-byte character was encountered. Perl considers this character to be the specified Unicode code point. Combining non-UTF-8 locales and Unicode is dangerous. Almost certainly some characters will have two different representations. For example, in the ISO 8859-7 (Greek) locale, the code point 0xC3 represents a Capital Gamma. But so also does 0x393. This will make string comparisons unreliable.

You likely need to figure out how this multi-byte character got mixed up with your single-byte locale (or perhaps you thought you had a UTF-8 locale, but Perl disagrees).

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 any of the codes @, /, U, u, w or a *-length. Redesign the template.

%s() with negative argument

(S misc) Certain operations make no sense with negative arguments. Warning is given and the operation is not done.

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%X’’ does not map to Unicode

(S utf8) 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 initialize the random seed for hashes, Perl could not get any randomness out of your system. This usually indicates Something Very Wrong.

Zero length \N{} in regex; marked by <--HERE in m/%s/

(F) Named Unicode character escapes (`\N{...}`) may return a zero-length sequence. Such an escape was used in an extended character class, i.e. `(?[...])`, or under `use re 'strict'`, which is not permitted. Check that the correct escape has been used, and the correct charnames handler is in scope. The <--HERE shows whereabouts in the regular expression the problem was discovered.

SEE ALSO

warnings, diagnostics.