

NAME

feature - Perl pragma to enable new features

SYNOPSIS

```
use feature qw(say switch);
given ($foo) {
    when (1)          { say "\$foo == 1" }
    when ([2,3])      { say "\$foo == 2 || \$foo == 3" }
    when (/^a[bc]d$/) { say "\$foo eq 'abd' || \$foo eq 'acd'" }
    when ($_ > 100)    { say "\$foo > 100" }
    default           { say "None of the above" }
}

use feature ':5.10'; # loads all features available in perl 5.10

use v5.10;           # implicitly loads :5.10 feature bundle
```

DESCRIPTION

It is usually impossible to add new syntax to Perl without breaking some existing programs. This pragma provides a way to minimize that risk. New syntactic constructs, or new semantic meanings to older constructs, can be enabled by `use feature 'foo'`, and will be parsed only when the appropriate feature pragma is in scope. (Nevertheless, the `CORE::` prefix provides access to all Perl keywords, regardless of this pragma.)

Lexical effect

Like other pragmas (`use strict`, for example), features have a lexical effect. `use feature qw(foo)` will only make the feature "foo" available from that point to the end of the enclosing block.

```
{
    use feature 'say';
    say "say is available here";
}
print "But not here.\n";
```

no feature

Features can also be turned off by using `no feature "foo"`. This too has lexical effect.

```
use feature 'say';
say "say is available here";
{
    no feature 'say';
    print "But not here.\n";
}
say "Yet it is here.";
```

`no feature` with no features specified will reset to the default group. To disable *all* features (an unusual request!) use `no feature ':all'`.

AVAILABLE FEATURES

The 'say' feature

`use feature 'say'` tells the compiler to enable the Perl 6 style `say` function.

See *"say" in perlfunc* for details.

This feature is available starting with Perl 5.10.

The 'state' feature

`use feature 'state'` tells the compiler to enable state variables.

See *"Persistent Private Variables" in perlsub* for details.

This feature is available starting with Perl 5.10.

The 'switch' feature

WARNING: Because the *smartmatch* operator is experimental, Perl will warn when you use this feature, unless you have explicitly disabled the warning:

```
no warnings "experimental::smartmatch";
```

`use feature 'switch'` tells the compiler to enable the Perl 6 given/when construct.

See *"Switch Statements" in perlsyn* for details.

This feature is available starting with Perl 5.10.

The 'unicode_strings' feature

`use feature 'unicode_strings'` tells the compiler to use Unicode rules in all string operations executed within its scope (unless they are also within the scope of either `use locale` or `use bytes`). The same applies to all regular expressions compiled within the scope, even if executed outside it. It does not change the internal representation of strings, but only how they are interpreted.

`no feature 'unicode_strings'` tells the compiler to use the traditional Perl rules wherein the native character set rules is used unless it is clear to Perl that Unicode is desired. This can lead to some surprises when the behavior suddenly changes. (See *"The 'Unicode Bug'" in perlunicode* for details.) For this reason, if you are potentially using Unicode in your program, the `use feature 'unicode_strings'` subpragma is **strongly** recommended.

This feature is available starting with Perl 5.12; was almost fully implemented in Perl 5.14; and extended in Perl 5.16 to cover `quotemeta`; and extended further in Perl 5.26 to cover *the range operator*.

The 'unicode_eval' and 'evalbytes' features

Together, these two features are intended to replace the legacy string `eval` function, which behaves problematically in some instances. They are available starting with Perl 5.16, and are enabled by default by a `use 5.16` or higher declaration.

`unicode_eval` changes the behavior of plain string `eval` to work more consistently, especially in the Unicode world. Certain (mis)behaviors couldn't be changed without breaking some things that had come to rely on them, so the feature can be enabled and disabled. Details are at *"Under the 'unicode_eval' feature" in perlfunc*.

`evalbytes` is like string `eval`, but operating on a byte stream that is not UTF-8 encoded. Details are at *"evalbytes EXPR" in perlfunc*. Without a `use feature 'evalbytes'` nor a `use v5.16` (or higher) declaration in the current scope, you can still access it by instead writing `CORE::evalbytes`.

The 'current_sub' feature

This provides the `__SUB__` token that returns a reference to the current subroutine or `undef` outside of a subroutine.

This feature is available starting with Perl 5.16.

The 'array_base' feature

This feature supports the legacy `$[` variable. See *"\$[" in perlvar* and *arybase*. It is on by default but disabled under `use v5.16` (see *IMPLICIT LOADING*, below).

This feature is available under this name starting with Perl 5.16. In previous versions, it was simply on all the time, and this pragma knew nothing about it.

The 'fc' feature

`use feature 'fc'` tells the compiler to enable the `fc` function, which implements Unicode casefolding.

See *"fc" in perlfunc* for details.

This feature is available from Perl 5.16 onwards.

The 'lexical_subs' feature

In Perl versions prior to 5.26, this feature enabled declaration of subroutines via `my sub foo, state sub foo` and `our sub foo` syntax. See *"Lexical Subroutines" in perlsub* for details.

This feature is available from Perl 5.18 onwards. From Perl 5.18 to 5.24, it was classed as experimental, and Perl emitted a warning for its usage, except when explicitly disabled:

```
no warnings "experimental::lexical_subs";
```

As of Perl 5.26, use of this feature no longer triggers a warning, though the `experimental::lexical_subs` warning category still exists (for compatibility with code that disables it). In addition, this syntax is not only no longer experimental, but it is enabled for all Perl code, regardless of what feature declarations are in scope.

The 'postderef' and 'postderef_qq' features

The 'postderef_qq' feature extends the applicability of *postfix dereference syntax* so that postfix array and scalar dereference are available in double-quotish interpolations. For example, it makes the following two statements equivalent:

```
my $s = "[@{ $h->{a} }]";
my $s = "[ $h->{a} ->@* ]";
```

This feature is available from Perl 5.20 onwards. In Perl 5.20 and 5.22, it was classed as experimental, and Perl emitted a warning for its usage, except when explicitly disabled:

```
no warnings "experimental::postderef";
```

As of Perl 5.24, use of this feature no longer triggers a warning, though the `experimental::postderef` warning category still exists (for compatibility with code that disables it).

The 'postderef' feature was used in Perl 5.20 and Perl 5.22 to enable postfix dereference syntax outside double-quotish interpolations. In those versions, using it triggered the `experimental::postderef` warning in the same way as the 'postderef_qq' feature did. As of Perl 5.24, this syntax is not only no longer experimental, but it is enabled for all Perl code, regardless of what feature declarations are in scope.

The 'signatures' feature

WARNING: This feature is still experimental and the implementation may change in future versions of Perl. For this reason, Perl will warn when you use the feature, unless you have explicitly disabled the warning:

```
no warnings "experimental::signatures";
```

This enables unpacking of subroutine arguments into lexical variables by syntax such as

```
sub foo ($left, $right) {  
    return $left + $right;  
}
```

See *"Signatures" in perlsub* for details.

This feature is available from Perl 5.20 onwards.

The 'refaliasing' feature

WARNING: This feature is still experimental and the implementation may change in future versions of Perl. For this reason, Perl will warn when you use the feature, unless you have explicitly disabled the warning:

```
no warnings "experimental::refaliasing";
```

This enables aliasing via assignment to references:

```
\$a = \$b; # $a and $b now point to the same scalar  
\@a = \@b; # to the same array  
\%a = \%b;  
\&a = \&b;  
foreach \%hash (@array_of_hash_refs) {  
    ...  
}
```

See *"Assigning to References" in perlref* for details.

This feature is available from Perl 5.22 onwards.

The 'bitwise' feature

WARNING: This feature is still experimental and the implementation may change in future versions of Perl. For this reason, Perl will warn when you use the feature, unless you have explicitly disabled the warning:

```
no warnings "experimental::bitwise";
```

This makes the four standard bitwise operators (& | ^ ~) treat their operands consistently as numbers, and introduces four new dotted operators (&. |. ^. ~.) that treat their operands consistently as strings. The same applies to the assignment variants (&= |= ^= &.= |.= ^.=).

See *"Bitwise String Operators" in perlop* for details.

This feature is available from Perl 5.22 onwards.

The 'declared_refs' feature

WARNING: This feature is still experimental and the implementation may change in future versions of Perl. For this reason, Perl will warn when you use the feature, unless you have explicitly disabled the warning:

```
no warnings "experimental::declared_refs";
```

This allows a reference to a variable to be declared with `my`, `state`, `our` or `localized` with `local`. It is intended mainly for use in conjunction with the "refaliasing" feature. See *"Declaring a Reference to a Variable" in perlref* for examples.

This feature is available from Perl 5.26 onwards.

FEATURE BUNDLES

It's possible to load multiple features together, using a *feature bundle*. The name of a feature bundle is prefixed with a colon, to distinguish it from an actual feature.

```
use feature ":5.10";
```

The following feature bundles are available:

bundle	features included
-----	-----
:default	array_base
:5.10	say state switch array_base
:5.12	say state switch unicode_strings array_base
:5.14	say state switch unicode_strings array_base
:5.16	say state switch unicode_strings unicode_eval evalbytes current_sub fc
:5.18	say state switch unicode_strings unicode_eval evalbytes current_sub fc
:5.20	say state switch unicode_strings unicode_eval evalbytes current_sub fc
:5.22	say state switch unicode_strings unicode_eval evalbytes current_sub fc
:5.24	say state switch unicode_strings unicode_eval evalbytes current_sub fc postderef_qq
:5.26	say state switch unicode_strings unicode_eval evalbytes current_sub fc postderef_qq

The `:default` bundle represents the feature set that is enabled before any `use feature` or `no feature` declaration.

Specifying sub-versions such as the 0 in 5.14.0 in feature bundles has no effect. Feature bundles are guaranteed to be the same for all sub-versions.

```
use feature ":5.14.0";    # same as ":5.14"
use feature ":5.14.1";    # same as ":5.14"
```

IMPLICIT LOADING

Instead of loading feature bundles by name, it is easier to let Perl do implicit loading of a feature bundle for you.

There are two ways to load the `feature` pragma implicitly:

- By using the `-E` switch on the Perl command-line instead of `-e`. That will enable the feature bundle for that version of Perl in the main compilation unit (that is, the one-liner that follows `-E`).
- By explicitly requiring a minimum Perl version number for your program, with the `use VERSION` construct. That is,

```
use v5.10.0;
```

will do an implicit

```
no feature ':all';  
use feature ':5.10';
```

and so on. Note how the trailing sub-version is automatically stripped from the version.

But to avoid portability warnings (see *"use" in perlfunc*), you may prefer:

```
use 5.010;
```

with the same effect.

If the required version is older than Perl 5.10, the `":default"` feature bundle is automatically loaded instead.