NAME

if - "use" a Perl module if a condition holds

SYNOPSIS

```
use if CONDITION, "MODULE", ARGUMENTS;
no  if CONDITION, "MODULE", ARGUMENTS;
```

DESCRIPTION

"use if"

The "if" module is used to conditionally load another module. The construct:

```
use if CONDITION, "MODULE", ARGUMENTS;
```

... will load "MODULE" only if "CONDITION" evaluates to true; it has no effect if "CONDITION" evaluates to false. (The module name, assuming it contains at least one "::", must be quoted when 'use strict "subs";' is in effect.) If the CONDITION does evaluate to true, then the above line has the same effect as:

```
use MODULE ARGUMENTS;
```

For example, the *Unicode::UCD* module's *charinfo* function will use two functions from *Unicode::Normalize* only if a certain condition is met:

```
use if defined &DynaLoader::boot_DynaLoader,
    "Unicode::Normalize" => qw(getCombinClass NFD);
```

Suppose you wanted "ARGUMENTS" to be an empty list, *i.e.*, to have the effect of:

```
use MODULE ();
```

You can't do this with the "if" pragma; however, you can achieve exactly this effect, at compile time, with:

```
BEGIN { require MODULE if CONDITION }
```

"no if"

The "no if" construct is mainly used to deactivate categories of warnings when those categories would produce superfluous output under specified versions of *perl*.

For example, the "redundant" category of warnings was introduced in Perl-5.22. This warning flags certain instances of superfluous arguments to "printf" and "sprintf". But if your code was running warnings-free on earlier versions of *perl* and you don't care about "redundant" warnings in more recent versions, you can call:

```
use warnings;
no if $] >= 5.022, q|warnings|, qw(redundant);

my $test    = { fmt  => "%s", args => [ qw( x y ) ] };
my $result  = sprintf $test->{fmt}, @{$test->{args}};
```

The "no if" construct assumes that a module or pragma has correctly implemented an "unimport()" method -- but most modules and pragmata have not. That explains why the "no if" construct is of limited applicability.

BUGS

The current implementation does not allow specification of the required version of the module.

SEE ALSO

Module::Requires can be used to conditionally load one or modules, with constraints based on the version of the module. Unlike "if" though, Module::Requires is not a core module.

Module::Load::Conditional provides a number of functions you can use to query what modules are available, and then load one or more of them at runtime.

The provide module from CPAN can be used to select one of several possible modules to load based on the version of Perl that is running.

AUTHOR

Ilya Zakharevich <mailto:ilyaz@cpan.org>.

COPYRIGHT AND LICENCE