NAME

re - Perl pragma to alter regular expression behaviour

SYNOPSIS

```
    use re 'taint';
    ($x) = ($^X =~ /^(.*)$/s);     # $x is tainted here

    $pat = '(?{ $foo = 1 })';
    use re 'eval';
    /foo${pat}bar/;     # won't fail (when not under -T switch)

    {
no re 'taint';     # the default
($x) = ($^X =~ /^(.*)$/s); # $x is not tainted here

no re 'eval';     # the default
/foo${pat}bar/;     # disallowed (with or without -T switch)
    }

    use re 'debug';     # NOT lexically scoped (as others are)
    /^(.*)$/s;      # output debugging info during
           #     compile and run time

    use re 'debugcolor';    # same as 'debug', but with colored output
    ...
```

(We use $^X in these examples because it's tainted by default.)

DESCRIPTION

When `use re 'taint'` is in effect, and a tainted string is the target of a regex, the regex memories (or values returned by the m// operator in list context) are tainted. This feature is useful when regex operations on tainted data aren't meant to extract safe substrings, but to perform other transformations.

When `use re 'eval'` is in effect, a regex is allowed to contain `(?{ ... })` zero-width assertions even if regular expression contains variable interpolation. That is normally disallowed, since it is a potential security risk. Note that this pragma is ignored when the regular expression is obtained from tainted data, i.e. evaluation is always disallowed with tainted regular expressions. See *"(?{ code })" in perlre*.

For the purpose of this pragma, interpolation of precompiled regular expressions (i.e., the result of `qr//`) is *not* considered variable interpolation. Thus:

```
    /foo${pat}bar/
```

is allowed if $pat is a precompiled regular expression, even if $pat contains `(?{ ... })` assertions.

When `use re 'debug'` is in effect, perl emits debugging messages when compiling and using regular expressions. The output is the same as that obtained by running a `-DDEBUGGING`-enabled perl interpreter with the **-Dr** switch. It may be quite voluminous depending on the complexity of the match. Using `debugcolor` instead of `debug` enables a form of output that can be used to get a colorful display on terminals that understand termcap color sequences. Set `$ENV{PERL_RE_TC}` to a comma-separated list of `termcap` properties to use for highlighting strings on/off, pre-point part on/off. See *"Debugging regular expressions" in perldebug* for additional info.

The directive `use re 'debug'` is *not lexically scoped*, as the other directives are. It has both compile-time and run-time effects.

See *"Pragmatic Modules" in perlmodlib*.