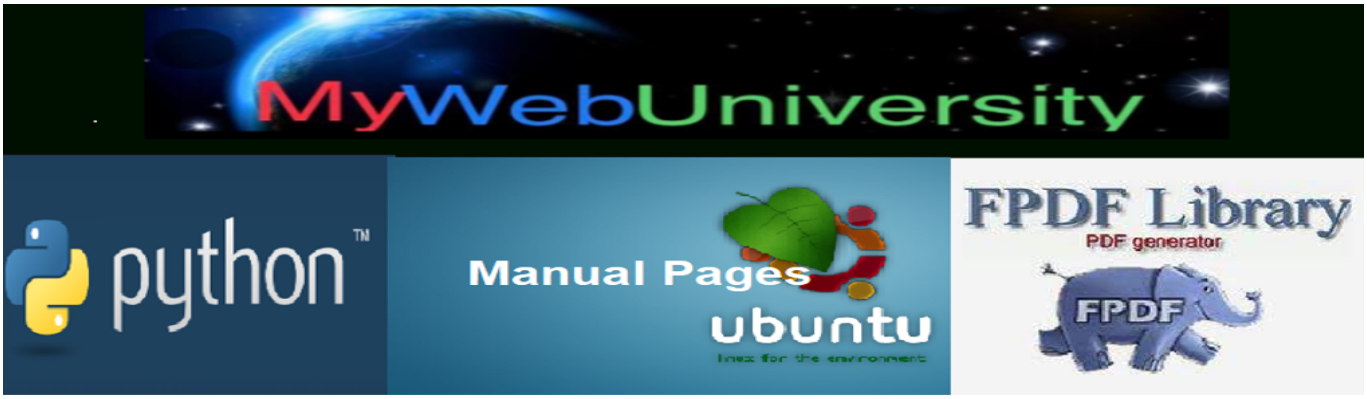

Rocky Enterprise Linux 9.2 Manual Pages on command 'strict.3perl'

$ man strict.3perl

strict(3perl) Perl Programmers Reference Guide strict(3perl)

NAME

strict - Perl pragma to restrict unsafe constructs

SYNOPSIS

```
use strict;

use strict "vars";
use strict "refs";
use strict "subs";

use strict;
no strict "vars";
```

DESCRIPTION

The "strict" pragma disables certain Perl expressions that could behave unexpectedly or are difficult to debug, turning them into errors. The effect of this pragma is limited to the current file or scope block.

If no import list is supplied, all possible restrictions are assumed. (This is the safest mode to operate in, but is sometimes too strict for casual programming.) Currently, there are three possible things to be strict about: "subs", "vars", and "refs".

"strict refs"

This generates a runtime error if you use symbolic references (see perlref).

```
use strict 'refs';
$ref = \$foo;
print $$ref;       # ok
$ref = "foo";
print $$ref;       # runtime error; normally ok
$file = "STDOUT";
print $file "Hi!";  # error; note: no comma after $file
```

There is one exception to this rule:

```
$bar = \&{'foo'};
&$bar;
```

is allowed so that "goto &$AUTOLOAD" would not break under stricture.

"strict vars"

This generates a compile-time error if you access a variable that was neither explicitly declared (using any of "my", "our", "state", or "use vars") nor fully qualified. (Because this is to avoid variable suicide problems and subtle dynamic scoping issues, a merely "local" variable isn't good enough.) See "my" in perlfunc, "our" in perlfunc, "state" in perlfunc, "local" in perlfunc, and vars.

```
use strict 'vars';
$X::foo = 1;         # ok, fully qualified
my $foo = 10;        # ok, my() var
local $baz = 9;      # blows up, $baz not declared before

package Cinna;
our $bar;                # Declares $bar in current package
```

```
    $bar = 'HgS';               # ok, global declared via pragma
```

The local() generated a compile-time error because you just touched a global name without fully qualifying it.

Because of their special use by sort(), the variables $a and $b are exempted from this check.

"strict subs"

This disables the poetry optimization, generating a compile-time error if you try to use a bareword identifier that's not a subroutine, unless it is a simple identifier (no colons) and that it appears in curly braces, on the left hand side of the "=>" symbol, or has the unary minus operator applied to it.

```
    use strict 'subs';
    $SIG{PIPE} = Plumber;   # blows up
    $SIG{PIPE} = "Plumber"; # fine: quoted string is always ok
    $SIG{PIPE} = \&Plumber; # preferred form
```

See "Pragmatic Modules" in perlmodlib.

HISTORY

"strict 'subs'", with Perl 5.6.1, erroneously permitted to use an unquoted compound identifier (e.g. "Foo::Bar") as a hash key (before "=>" or inside curlies), but without forcing it always to a literal string.

Starting with Perl 5.8.1 strict is strict about its restrictions: if unknown restrictions are used, the strict pragma will abort with

```
  Unknown 'strict' tag(s) '...'
```

As of version 1.04 (Perl 5.10), strict verifies that it is used as "strict" to avoid the dreaded Strict trap on case insensitive file systems.